Weed detection in soybean crop field using CNN

A Thesis Presented

By

Abiy Simon Girumu

To

The Faculty of Informatics
of

St. Mary's University

in

Partial Fulfillment of the Requirements

for the Degree of Master of Science

In

Computer Science

February, 2024

ACCEPTANCE

PLANT SPECIES CLASSIFICATION USING DEEP LEARNING

By

Abiy Simon Girumu

Accepted by the Faculty of Informatics, St. Mary's University, in partial fulfillment of the requirements

For the degree of Master of Science in Computer Science

Thesis Examination Committee:

Internal Examiner

{Alembante Mulu (Ph.D.), Feb 2024}

External Examiner

{Minale Ashagre (Ph.D.), Feb 2024}

Dean, Faculty of Informatics

{Alembante Mulu (Ph.D.), Feb 2024}

{Date of Defense}

February 2024

DECLARATION

I, the undersigned, declare that this thesis work is my original work, has not been presented for a

Degree in this or any other universities, and all sources of materials used for the thesis work have

been duly acknowledged.

Abiy Simon Girumu

____

Signature

Addis Ababa

Ethiopia

This thesis has been submitted for examination with my approval as advisor.

Million Meshesha (Ph.D.)

____

Signature

Addis Ababa

Ethiopia

{Exact Date of Defense}

{February 2024}

Acknowledgment

First and foremost, I would like to express my deepest gratitude to the Almighty God for providing the strength, wisdom, and resilience needed to navigate the challenges encountered during this research journey. The divine guidance has been a source of inspiration and motivation.

I would like to express my deepest gratitude to my advisor, Dr. Million Meshesha, for his unwavering support, invaluable guidance, and mentorship throughout the course of this research. His expertise and encouragement have been instrumental in shaping the trajectory of this work.

A heartfelt thank you goes to my family, who have been a constant source of support and encouragement. In particular, I want to express my deepest appreciation to my grandmother, Etenesh Birhanu, and my mom, Arenchata Terefe, for their unwavering love, encouragement, and sacrifices. Their belief in my abilities has been a driving force, and I am truly fortunate to have them as pillars of support.

To all those who have provided support, encouragement, and understanding, I extend my sincere thanks. Your contributions, whether big or small, have played a crucial role in the successful completion of this research.

Table of contents

List of figures

List of tables

List of Acronym

ANN	ARTIFICIAL NEURAL NETWORK
CNN	CONVENTIONAL NEURAL NETWORK
DIP	DIGITAL IMAGE PROCESSING
DL	DEEP LEARNING
FCL	FULLY CONNECTED LAYER
FCN	FULLY CONVOLUTIONAL NETWORK
CLAHE	CONTRAST LIMITED ADAPTIVE HISTOGRAM EQUALIZATION
ReLu	RECTIFY LINEAR UNIT
KNN	K-NEAREST NEIGHBORS
DRCNN	DEEP RESIDUAL CONVOLUTIONAL NEURAL NETWORK
HOG	HISTOGRAM OF ORIENTED GRADIENTS
GLCM	GRAY-LEVEL CO-OCCURRENCE MATRIX
SLIC	SIMPLE LINEAR ITERATIVE CLUSTERING

Abstract

This research contributes towards addressing the significant challenge of manual labor in weed detection and emphasizes the need for the development of an efficient system using digital image processing and deep learning techniques. The primary aim is to detect broadleaf weeds in soybean crops, leveraging the capabilities of Convolutional Neural Networks (CNNs) for image representation and pattern recognition. The study adopts an experimental research methodology, collecting a substantial dataset comprising 7,000 soybean images and 17,000 images for broadleaf weed, soil, and grass classes. To ensure balanced training, 2,000 images from each class are used.

Digital image preprocessing is used to get the data gathered ready for analysis. The acquired image data passes through cleaning and data filtering. Additionally, in order to give class labels and identify regions of interest for crop and weed identification, relevant labelling and annotation processes are carried out.

The classification using convolutional neural network (CNN) involves four classes: broadleaf weed, soil, soybean, and grass. Utilizing 80% of the total dataset for training and the remaining 20% for testing, the experimental results demonstrate the efficiency and accuracy of the proposed model. Specifically, the model achieves a weed detection accuracy of 94.46%, indicating its promising potential for real-time weed detection in soybean fields. This accomplishment is a crucial step towards mitigating the reliance on manual labor, enabling timely and accurate weed identification to enhance crop management.

Looking ahead, future work is recommended to expand the model's recognition capabilities to include other weed species and roots. The incorporation of a larger dataset with diverse images would further enhance the model's robustness and generalization to different environmental conditions. The ultimate goal is to develop a comprehensive and versatile model that can contribute significantly to precision agriculture by not only identifying broadleaf weeds but also expanding its scope to encompass various weed types and agricultural challenges.

Keywords: Weed Detection; Soybean Crop Field; Digital Image Processing (DIP); Convolutional Neural Network (CNN)

CHAPTER ONE

INTRODUCTION

1.1. Background of the study

Ethiopia's agriculture sector is extremely important, contributing significantly to the country's economy and the livelihoods of its people. Ethiopia, located in the Horn of Africa, has distinct agro-ecological zones that support a wide variety of agricultural activities. The sector employs a sizable proportion of the workforce and contributes to food security, export revenues, and rural development. However, the sector confronts a number of obstacles that limit its ability to meet rising demand for agricultural products, notably in terms of weed management [1].

According to a study conducted by Amsalu (2019) for a large section of Ethiopia's population, agriculture is the primary source of work and income. Agriculture provides a living for more than 70% of Ethiopia's population [1]. This high level of dependence underlines the sector's important role in creating job opportunities and contributing to rural development.

Due to a lack of modern farm technology and equipment, agricultural practices in Ethiopia are mostly carried out manually. Smallholder farmers make up the vast majority of the agricultural workforce and rely significantly on physical labor for duties such as land preparation, sowing, weeding, and harvesting. This reliance on manual labor raises concerns about efficiency, productivity, and the ability to handle labor-intensive chores like weed management [2].

Weeds are one of the most serious issues that Ethiopian farmers confront, affecting agricultural production and crop harvests [3]. Weeds compete with crops for resources like as water, nutrients, and sunlight, causing crop growth to slow and yield losses to occur. Ethiopia struggles to produce enough food due to both external and internal problems, particularly in efficiently managing weeds.

Identifying crops and weeds is a crucial field of agricultural study since it can significantly affect crop output, quality, and profitability [3]. Informed judgments about crop management, such as deciding how much fertilizer or herbicide to apply and identifying areas that need extra care or intervention, can be made by farmers with the ability to accurately and effectively distinguish between crops and weeds. Deep learning algorithms have recently gained popularity as a promising method for identifying crops and weeds because they can reach high levels of accuracy by utilizing vast amounts of data and intricate neural network structures.

External variables such as climate variability provide problems to Ethiopian agricultural productivity. The country's agro-climatic characteristics range from desert and semi-arid regions to highland regions. Crop failure and weed infestations can be exacerbated by unpredictable rainfall patterns, hard droughts, and unpredictable weather events. These climatic constraints have an impact on the availability of water for irrigation, planting timing, and total crop development [4].

Internal variables within the agricultural sector also contribute to weed management challenges. Smallholder farmers confront considerable challenges due to limited access to resources. This includes a shortage of high-quality seeds, fertilizers, insecticides, and herbicides, as well as insufficient water supply [3]. Inadequate access to these resources blocks efficient weed management practices and decreases agricultural productivity overall.

Furthermore, farmers' lack of knowledge and abilities in weed identification and management approaches contributes to the weed control difficulty. This problem gets worse by farmers' lack of access to agricultural extension services, which offer them with information, training, and technical assistance. Weed control efforts are unsatisfactory due to a lack of information about efficient weed management practices and the right use of herbicides.

To address these challenges and improve weed management in Ethiopia, research and technological advancements play a crucial role.

To summarize, Ethiopia's agriculture industry is critical to the country's economy and the livelihoods of its people. A sizable section of the population relies on agriculture for a living, with manual labor being the most common practice. Weed management presents considerable obstacles, which are influenced by external factors such as climatic variability as well as internal issues such as restricted access to resources and insufficient knowledge and skills. Addressing these issues is critical for increasing agricultural output and ensuring Ethiopia's food security. The country can improve weed control practices and strengthen agriculture's contribution to sustainable development through research, technical developments, and technological advancements.

So, in this research an attempt is made to apply digital image processing and deep learning for identifying weeds from soybean crop images using deep learning.

1.2. Deep learning

There are various machine learning algorithms used for image classification, including Support Vector Machines (SVM), Random Forests, Convolutional Neural Networks (CNN), and Deep Learning models. These algorithms can handle complex feature representations and learn hierarchical patterns in the image data.

Deep learning has emerged as a powerful paradigm in machine learning due to its ability to automatically learn hierarchical representations from data. Unlike traditional machine learning approaches that often require manual feature engineering, deep learning models can autonomously discover and utilize relevant features from raw input. The strength of deep learning lies in its capacity for end-to-end learning, enabling models to directly map input to output without the need for explicit intermediate steps. This characteristic simplifies system architectures and streamlines the development process. Moreover, deep learning thrives on large datasets, leveraging scale and big data to improve model performance. Its effectiveness in capturing complex non-linear relationships in data makes it suitable for diverse applications such as image recognition, natural language processing, and speech recognition. Transfer learning further enhances its utility by allowing pre-trained models to be fine-tuned for specific tasks, enabling efficient knowledge transfer. The versatility of deep learning, demonstrated across various domains, has solidified its position as a key technology in contemporary artificial intelligence [5,6,7].

1.3. Motivation of the study

Weeds are undesired plants that compete against productive crops for space, light, water, and soil nutrients and propagate themselves either through seeding or rhizomes. They are generally poisonous, produce thorns and burrs, and hamper crop management by contaminating crop harvests. Smaller weed seedlings with a slow growth rate are more difficult to detect and manage than larger ones which grow vigorously. Weed management is complicated because the competitive nature of weeds can vary in different conditions and seasons [8].

To overcome the difficulties and restrictions of conventional methods in agriculture, crop and weed detection using machine learning and deep learning is required [8]. Conventional methods for identifying crops and weeds frequently rely on human labor, visual inspection, and chemical analysis, which are labor intensive, error-prone, and time-consuming. These restrictions reduce the effectiveness and output of farmers and present substantial difficulties for sustainable agricultural practices.

The main goal of this research is to bypass the shortcomings of traditional methods and give farmers a trustworthy and impartial tool for crop and weed identification. Outstanding performance in a range of computer vision tasks, including object detection and image classification, has been shown using deep learning systems. We can take advantage of deep learning's capacity to automatically extract useful characteristics and accurately categorize crops and weeds by training a CNN model on a large and representative collection of crop and weed digital images.

In conclusion, the goal of this thesis is to investigate and utilize the potential of computer vision's deep learning algorithms for crop and weed detection. Farmers can gain from objective crop and weed identification by creating a system that is completely based on computer vision analysis. This would enable more effective resource allocation, better decision-making, and improved agricultural practices. With the potential to completely transform crop management systems, this research advances computer vision techniques in agriculture and gives farmers the tools they need to increase production and cut expenses.

1.4. Statement of the Problem

The requirement for an effective and precise system for crop and weed detection utilizing deep learning techniques in the field of computer vision are the issue this thesis attempts to investigate. Agriculture's traditional methods for identifying crops and weeds frequently rely on labor-intensive human labor and subjective visual assessment, which can be inaccurate and time-consuming. Convolutional neural networks (CNNs), in particular, have the potential to be used to construct a crop and weed detection system that is entirely computer vision-based, obviating the need for human participation.

Although deep learning-based techniques for crop and weed detection have demonstrated encouraging results, there are still issues that need to be resolved. Deep learning-based methods for weed and crop identification have been widely studied in the past, and the results have shown both the improvements and the problems still facing the field. The following are some of the major issues these studies looked into, along with the noted research gaps.

Prior studies have concentrated on enhancing accuracy of weed identification programs. For example, as noted by, Ferreira et al [9]. And Razfar et al. [10] have shown that deep learning works well in this situation. Still, there is a need for more research to close the false positives gap—the problem of misclassifying non-weed objects as weeds. It is imperative to refine models in order to reduce false alarms before implementing them in agriculture.

The detection process's performance of deep learning can be considerably impacted by changes in illumination, occlusions, and the presence of plants that look identical to each other [9]. A further obstacle to establish accurate and robust detection performance is the absence of broad and representative datasets for training deep learning models specifically for crop and weed identification.

The aim of this thesis is to create an effective and precise system for crop and weed detection in agricultural fields that only uses computer vision techniques, notably deep learning. The system must be capable to manage variations in ambient circumstances, plant features, and any confounding factors in addition to overcoming the limits of conventional methods accurately classifying crops and weeds based on visual analysis.

1.5. Research Questions

In order to investigate the above stated problem, this study formulates the following research questions.

1. What image processing techniques can be employed to preprocess image datasets effectively for the accurate classification of crops and weeds in soybean fields?

2. Which deep learning architectures demonstrate optimal performance in distinguishing between crops and weeds within agricultural landscapes, particularly in soybean fields?
3. How does the proposed deep learning model perform in accurately identifying and delineating weeds amidst crop vegetation in soybean fields, and what are its strengths and limitations in this context?

1.6. Objective of the study

The general objective of this study is to construct an optimal model for crop and weed detection using digital image processing and deep learning.

1.7. Specific Objectives

1. To conduct a comprehensive review of the state-of-the-art CNN models, focusing on their applicability and performance in agricultural image analysis, specifically in the context of crop and weed detection in soybean fields.

2. To prepare representative and diverse datasets, considering variations in environmental conditions, crop species, and weed types

3. To investigate and identify suitable deep learning architectures and image processing techniques for experimentation.

4. To train and optimize a deep learning model using, to achieve high accuracy and robustness in crop and weed detection.

5. To evaluate the effectiveness of the proposed crops and weed detection

1.8. Scope and limitations of the study

The goal of the research is to employ deep learning methods in the field of computer vision to distinguish between soybean crops and weeds. The goal of the study is to create a precise and effective system that can categorize and discriminate between weeds and soybean crops based solely on visual analysis. Convolutional neural networks (CNNs) and image processing methods are mostly used to accomplish this.

It is important to remember that this research is restricted to the identification and separation of crops and weeds. Additional aspects of crop management, such as identifying diseases, predicting yields, or evaluating the health of plants, are not included. In order to help farmers make sound decisions about the distribution of resources and focused treatments, the research is specifically designed to address the problem of crop and weed identification.

Additionally, this study does not use automation tools for weed and crop detection, such as drones or robots. Deep learning algorithms are only used to analyze computer vision data. To increase the proposed system's practical usability, the addition of automation systems can be explored as a viable field for future research.

The study also admits the drawbacks of the diversity and accessibility of datasets used to train the deep learning model. Although every effort has been taken to maintain representativeness and generalizability, the challenge of small datasets may have an impact on the overall performance and generalizability of the produced system.

The performance evaluation of the proposed system is based on publicly available datasets, which is also used to conduct the research. However, it is acknowledged that the traits and environmental circumstances of particular agricultural locations might change, which may affect the system's performance in the actual world. Therefore, the specific context and implementation factors should be taken into account while interpreting the research findings.

Future studies could investigate the integration of automation technologies, such as drones or robots, to improve the efficiency and practicability of crop and weed detection. Additional study should concentrate on enhancing the system's efficacy by gathering and utilizing more varied and representative datasets, such as those particular to various crop species and weed types.

1.9. Significance of the study (Contributions)

For a number of stakeholders in the agriculture industry, the study on crop and weed detection applying deep learning techniques in the field of computer vision is very important. The conclusions and results of this study have the potential to result in the following important advantages and advancements:

Increased Accuracy and Efficiency: The created system has the potential to dramatically increase the accuracy and efficiency of crop and weed detection by utilizing deep learning techniques. Farmers may be able to make fast and accurate decisions as a result, which help them spend resources wisely, improve their crop management techniques, and carry out focused interventions as needed.

Reduced Manual Labor: Traditional techniques for identifying crops and weeds frequently involve a lot of manual labor and visual inspection, which can be tedious, arbitrary, and vulnerable to mistakes. The suggested solution, which uses computer vision and deep learning to distinguish between crops and weeds, lessens the need for physical labor and offers an automated and objective method. Farmers may be able to focus on other important farming duties as a result of this, saving them vital time and resources.

More Sustainable Agriculture practices: Accurate and effective crop and weed identification can help foster more sustainable agricultural methods. Farmers can undertake targeted, localized interventions that reduce their reliance on chemical herbicides and support environmentally friendly practices by accurately detecting weeds. This may result in fewer adverse effects on the environment, less use of chemicals, and increased sustainability in general for agricultural operations.

Cost Savings: Farmers may be able to save money using the suggested system. Farmers may use resources like fertilizers, irrigation, and herbicides more efficiently by properly detecting and distinguishing between crops and weeds. This focused strategy lowers the chance of wasteful spending and agricultural production loss from weed competition.

Advancement of Computer Vision Techniques in Agriculture: The work advances computer vision techniques in agriculture, particularly in the context of crop and weed detection. The research demonstrates the potential of modern technology to overcome significant issues in precision agriculture by utilizing deep learning algorithms and image processing techniques. The findings of this study may stimulate additional investigation and development in the area, resulting in ongoing improvements in agricultural technology.

In conclusion, this study's significance lies in its potential to increase the effectiveness and precision of crop and weed detection, decrease manual labor, support sustainable agricultural practices, achieve cost savings, contribute to technological advancement in precision agriculture, and broaden the body of knowledge in the area of computer vision and deep learning in agriculture. The agricultural industry may experience significant changes as a result of their contributions, which would be advantageous to farmers, the environment, and larger food production systems.

The significance of the study outlined in the provided text has broader implications not only for stakeholders in the agriculture industry but also for researchers in the field of computer vision and deep learning. Here's how the significance of the study extends to researchers:

Advancement of Research in Computer Vision and Deep Learning: This study contributes to the ongoing advancement of research in computer vision and deep learning, specifically within the context of crop and weed detection. By demonstrating the potential of modern technology to address significant challenges in precision agriculture through the utilization of deep learning algorithms and image processing techniques, this research sets a foundation for further exploration and development in these domains. Researchers can leverage these insights to build upon existing methodologies, propose innovative solutions, and contribute to the evolving landscape of agricultural technology.

Stimulating Further Investigation: The findings of this study have the potential to stimulate additional research and investigations in the field. By showcasing the effectiveness of deep learning techniques in addressing real-world challenges in agriculture, researchers may be inspired to explore related issues, refine existing algorithms, or develop novel approaches. This study acts as a catalyst for further exploration and experimentation in the application of computer vision and deep learning to various agricultural problems, fostering a culture of continuous improvement and innovation.

Contribution to the Body of Knowledge: The study significantly contributes to the body of knowledge in the intersection of computer vision, deep learning, and agriculture. Researchers can benefit from the insights, methodologies, and results presented in this study as a reference point for their own work. It enriches the existing literature by providing practical applications of deep learning in the agricultural sector, potentially guiding future research endeavors and inspiring collaborative efforts among researchers working on similar topics.

In summary, the significance of the study for researchers lies in its potential to inspire further advancements, stimulate new investigations, and contribute valuable insights to the growing body of knowledge in computer vision, deep learning, and their applications in agriculture. Researchers can build upon this foundation to explore diverse aspects of precision agriculture and contribute to the continuous evolution of technology in this field.

1.10. Methodology of the study

1.10.1. Research design

In order to determine whether deep learning approaches are useful for crop and weed detection, this study uses an experimental research methodology. A scientific method called experimental research involves changing factors and seeing how those changes affect desired outcomes. In order to get trustworthy results, it enables researchers to identify cause-and-effect links between variables [9].

In the context of this study, controlled experiments are designed and carried out in order to assess the effectiveness and performance of deep learning models for crop and weed detection. The following essential steps are part of the study design [11, 12]:

To conduct an extensive experimentation, three major steps are considered, such as data collection and preparation, implementation and evaluation for achieving the objective of the study.

1.10.2. Data collection and preparation

Internet databases that offer publicly accessible datasets are crucial to crop and weed detection, like Google Datasets or GitHub, is the source of the data for this study. Numerous datasets have been uploaded to these platforms by scholars and organizations from around the world. The information gathered from these sources is carefully chosen based on their applicability to the study's goals and the availability of crop and weed photographs with annotations [13, 14].

To consider the context in some of the places within Ethiopia, image captured by using digital camera with 16 megapixels. The selection of the acquiring tool is done considering elements such as image resolution, portability, and compatibility with the environment used for gathering field data.

Preprocessing procedures is used to get the data gathered or obtained from internet sources ready for analysis. We'll use the next few data preprocessing steps [13, 14]:

Cleaning the data: The acquired data may have noise, artefacts, or insufficient annotations. To clean up the datasets of any errors, inconsistencies, or unnecessary data, data cleaning procedures is used.

Data Filtering and Annotation: The data gathered may include a variety of photos, including ones unrelated to weeds or crops. Only the pertinent photographs chosen and kept for the study using data filtering procedures. Additionally, in order to give class labels and identify regions of interest for crop and weed identification, relevant labelling and annotation processes is carried out if the acquired data lacks annotations.

Data Augmentation: To make the dataset larger and more diverse, data augmentation techniques used throughout the research. To create more training examples, methods including picture rotation, flipping, scaling, and the inclusion of artificial variances is used. The generalization and robustness of the deep learning models after training are improved by data augmentation.

1.10.3. Implementation Tools

The best software application for crop and weed detection using CNN depends on a number of variables, including the study's specific objectives, the user's level of knowledge, and compatibility with existing systems. There isn't an unambiguous "best" tool because each one offers advantages and qualities of its own. But among the most well-liked and effective frameworks for deep learning applications, such as crop and weed detection, are TensorFlow and PyTorch but we can list some of other well-known tools as follows:

There are several software tools that are commonly used for crop and weed identification through Convolutional Neural Networks (CNN). Let's explore some of these tools and provide a brief description of each:

TensorFlow: is an open-source machine learning framework developed by Google. It provides a comprehensive ecosystem for building and deploying CNN models. Tensor Flow offers high-level APIs, such as Keras, which simplifies the development process. It supports distributed training, model serving, and deployment on various platforms.

PyTorch: is another popular open-source deep learning framework. It provides dynamic computational graphs and a Pythonic interface, making it easy to build and train CNN models. PyTorch offers extensive support for GPU acceleration and has a vibrant community with a wealth of pre-trained models and research advancements.

Keras: is a high-level neural networks API written in Python. It runs on top of other deep learning frameworks, including TensorFlow and Theano. Keras simplifies the process of building and training CNN models by providing a user-friendly and intuitive interface. It allows rapid prototyping and has extensive documentation and community support.

OpenCV: (Open-Source Computer Vision Library) is a widely used open-source computer vision library. While it is not specifically a deep learning framework, OpenCV provides a rich set of image processing and computer vision algorithms that can be used in conjunction with CNN models. It offers functionalities such as image loading, preprocessing, and visualization.

The capabilities, benefits, and level of community assistance offered by each of these software applications vary for CNN-based crop and weed detection. The selection of a tool is influenced by various elements, including personal preference, study requirements, resource availability, and compatibility with already-in-use frameworks or systems. When choosing a software solution for crop and weed identification using CNN, it is crucial to take the documentation, community support, and ease of interface with other libraries or hardware into account.

Alongside with python which is the programing language that is chosen for proceeding this study. Python is widely used for image processing due to several reasons which is ease of use, large libraries and ecosystem, integration with other technologies, active community and support, performance optimization

CHAPTER TWO

LITERATURE REVIEW

This chapter explores the complex interaction between weeds and crops in agricultural environments. It examines the significant effects of weed growth on farms and talks about different methods of detection which utilize the use of deep learning and digital image processing. The chapter also explores the categorization of various weed types, which adds to our understanding of this dynamic interaction in agricultural environments.

2.1. Agricultural Sector

An important factor in the growth of the world economy has been agriculture. It is essential to human existence because it provides sustenance. Four percent of the world's total economic productivity came from agriculture. Moreover, it provides a means of livelihood for one billion individuals globally, serves as a supply of raw materials for many businesses, and plays a crucial part in managing global environmental conditions. [15].

The agricultural sector is at the heart of the economies of the least developed countries (LDCs). It accounts for a large share of gross domestic product (ranging from 30 to 60%), employs a large proportion of the labor force (from 40% to 90%), represents a major source of foreign exchange (from 25% to 95%), supplies the bulk of basic food and provides subsistence and other income to more than half of the LDCs' population [15].

Agriculture is the backbone of the Ethiopian economy and it contributes about 50% of the country's gross domestic product (GDP) and more than 80% of its exports. Furthermore, it is one of the main employment sectors with about 80% of the country's population depending on the agricultural sector for their livelihoods. The agricultural sector of Ethiopia is dominated by smallholder farming. In Ethiopia, about 95% of main crops (e.g., cereals, pulses, oilseeds, vegetables, root crops, fruits, and cash crops) are produced by smallholder farms [16].

However, these farms are facing various constraints that hamper crop productivity. Major constraints include poor soil fertility, severe land degradation, high dependence on rainfall, low availability and poor quality of seeds and fertilizers, economic constraints like low income, lack of financial support, waterlogging in wetland areas, salinity in arid and semi-arid areas, acidity in high rainfall areas, pests (like weeds, diseases, and insects) [16, 17].

2.2. Weed

According to definitions, a "weed" is a plant that is undesirable, out of place, or a problem because it gets in the way of crops or animal production. Generally speaking, the phrase refers to any plant species that frequently becomes problematic. [18].

Weeds are unwanted plants that compete with cultivated crops for resources, affecting agriculture negatively. They interfere with crop growth by competing for sunlight, water, and nutrients, leading to reduced crop yields and economic losses. Weeds can also host pests and diseases, further impacting crop health and yield [18].

Weed infestations result in reduced crop quality and market value. They hinder efficient farm operations by necessitating additional labor and resources for weed control. Herbicide resistance is a growing concern, affecting weed management effectiveness and necessitating innovative approaches [18]

Weeds impact other crops by serving as hosts for pests and diseases that can transfer to cultivated plants. Additionally, weeds can harbor pathogens that affect crop health. Controlling weeds is crucial for sustainable agriculture, enhancing crop competitiveness and minimizing yield losses. Climate-smart agriculture practices have been explored to manage weed density and diversity, emphasizing integrated strategies like zero-till, crop rotation, residue retention, and irrigation [19].

2.2.1. Pathways of Weed Spread to Crop Fields

Weeds are unwanted plants that compete with crops for water, nutrients, light, and space. They can reduce crop yields and quality and increase production costs. Weeds can spread to crop fields in a variety of ways, including the following [20]:

Seeds - Weeds produce many seeds, which can be dispersed by wind, water, animals, and humans. Wind-dispersed seeds can travel long distances, and water-dispersed seeds can float downstream to new areas. Animals can spread weed seeds by eating and then dispersing the seeds in their droppings. Humans can spread weed seeds by transporting contaminated seed, feed, and machinery [20].

Weed seed can stay viable in the soil for many years, and both vegetative tissue and weed seed can spread over long distances to infest new fields. The weed seedbank in any particular place is home to an extensive collection of weed species and ecotypes that are ready to germinate when the right signal is received and that are adaptable to a wide variety of environmental circumstances. According to a research, there were anywhere between 98 and 3,068 viable weed seeds in a square foot of soil that was 6 inches thick. This indicates that every acre, there are between 4.3 million to 133 million viable seeds. [21].

Vegetative fragments - Some weeds can spread by vegetative fragments, such as root pieces, stems, and leaves. These fragments can be transported by water, machinery, or human activity. When vegetative fragments are deposited in a new area, they can sprout and grow into new plants [22].

Contaminated manure - Manure can contain weed seeds, which can spread to crop fields when manure is applied as fertilizer [23].

Equipment and machinery - Farm equipment and machinery can transport weed seeds and other propagules from one field to another [23].

Irrigation water - Irrigation water can contain weed seeds and other propagules, which can spread to crop fields when the water is applied [24].

Human activity - Humans can spread weed seeds to crop fields by walking through fields, carrying contaminated clothing and equipment, and transporting contaminated seed, feed, and manure [25].

Conclusion

Weeds can spread to crop fields in a variety of ways, including by seeds, vegetative fragments, contaminated manure, irrigation water, equipment and machinery, and human activity. It is important to be aware of these pathways of spread in order to develop effective weed control strategies.

2.2.2. Types of weeds

According to the Global Biodiversity Information Facility (GBIF), there are over 400,000 species of vascular plants that are considered weeds. This number is likely to be higher, as not all weed species have been identified and classified [26].

Weeds can be classified in a variety of ways, such as by their life cycle (annual, biennial, or perennial), growth form (broadleaf, grassy, or sedge), or habitat (terrestrial, aquatic, or wetland).

Here are some common types of weeds [26]:

Broadleaf weeds: These weeds have broad leaves, such as dandelions, clover, and chickweed.

Grassy weeds: These weeds have narrow, grass-like leaves, such as crabgrass, Bermuda grass, and quack grass.

Sedge weeds: These weeds have three-sided stems and narrow leaves, such as nutsedge and horsetail.

Aquatic weeds: These weeds grow in water, such as duckweed, water hyacinth, and hydrilla.

In terms of danger, some weeds are more harmful than others due to their invasive nature and ability to outcompete crops. Weeds like Palmer amaranth (Amaranthus palmeri) and waterhemp (Amaranthus tuberculatus) are particularly aggressive and resistant to herbicides, posing significant threats to agriculture.Certain weeds closely resemble crops, making their identification challenging. False cleavers (Galium spurium) can resemble crops like soybean due to similar leaf shapes, potentially leading to mismanagement.

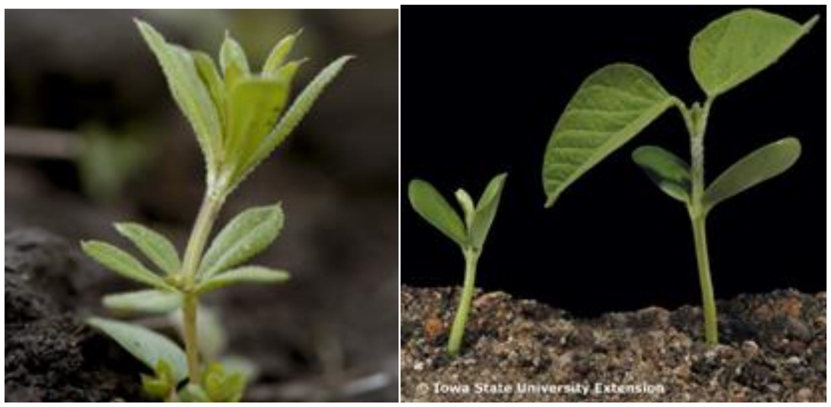


Fig 2- 1 Resemblance of Galilum spurium with soybean at early stage of growing [27]

2.3. Soybean

Soybean is one of the most important crops in Ethiopia. It contributes nearly 18% to the country's total oilseed production and accounts for only 6% of the area planted to oilseeds. The area allocation for soybean, and its production, as well as productivity, are increasing under smallholder farmers in Ethiopia. The crop is a multiple-purpose crop; an important source of protein and raw material for industries to produce food, feed, and oil. The USDA predicts that the production of soybean will increase in Ethiopia, partially due to requirements for the raw material in oil, livestock feed soybean meal for poultry production. Ethiopia's oilseed sector plays an important role in generating foreign exchange earnings. Oilseed crops are the third largest foreign exchange-earners, next to coffee and cut flowers [28].

Due to a variety of variables and interactions in agricultural ecosystems, several weed species frequently thrive near soybean crops. Among these are the soil characteristics. Specific weed species may flourish in soybean fields according to the type of soil there. Weeds are more likely to flourish in soil that is suitable for them in terms of pH, texture, and nutrient concentration [29].

Top 7 Weeds in Soybean (62 survey respondents)			
MOST COMMON		MOST TROUBLESOME	
1	water hemp	1	Palmer amaranth
2	common lambs quarters	2	water hemp
3	foxtail spp.	3	ragweed spp.
4	Palmer amaranth	4	horseweed (marestail)
5	horseweed (marestail)	5	Morningglories spp.
6	morning-glory spp.	6	common lambs quarters
7	redroot pigweed	7	kochia

Table 2- 1 Composite List of Weeds is used for weed around soybean crop [30]

Here is information about the behavior and characteristics of each of the mentioned most aggressive weed types [31,32,33,34]:

2.3.1. Palmer Amaranth:

Behavior: Palmer amaranth (Amaranthus palmeri) is a highly invasive and aggressive weed species. It's known for its rapid growth, competitive nature, and adaptability to various environments.

Characteristics: It can reach heights of up to 8 feet, making it a formidable competitor with crops for sunlight, nutrients, and water. Palmer amaranth is particularly problematic because it has developed resistance to multiple herbicides, making control challenging [31].

2.3.2. Water Hemp:

Behavior: Waterhemp (Amaranthus tuberculatus) is another aggressive weed species, especially in row crops like soybeans and corn. It can quickly become herbicide-resistant, making it challenging to manage.

Characteristics: Waterhemp can grow up to 9 feet tall and produces thousands of seeds per plant. It competes with crops for resources and can significantly reduce yields if not controlled effectively [32].

2.3.3. Ragweed spp.:

Behavior: Ragweed species, including common ragweed (Ambrosia artemisiifolia), are known for causing allergies due to their prolific pollen production. They can quickly colonize disturbed areas.

Characteristics: Ragweed has deeply lobed leaves and produces an abundance of small, inconspicuous flowers. It spreads through wind-dispersed pollen and seeds, often affecting individuals with allergies [33].

2.3.4. Horseweed (Marestail):

Behavior: Horseweed (Conyza canadensis) is an adaptable weed that can grow in a variety of **conditions**. It's known for its resistance to glyphosate herbicides [33].

Characteristics: It has a rosette-like basal growth form and can grow tall, with slender stems and small, white flowers. Marestail can quickly spread in fields and outcompete crops [33].

2.3.5. Morningglories.:

- Behavior: Morningglories species are climbing and trailing vines known for their heart-shaped leaves and trumpet-shaped flowers. They can become invasive in agricultural and natural settings.

Characteristics: These weeds have twining stems and showy flowers in various colors. They can smother crops and other plants if not managed effectively [33].

2.3.6. Common Lambsquarters:

- Behavior: Common lambsquarters (Chenopodium album) is a common weed in agricultural fields. It competes with crops for nutrients and can reduce yields.

Characteristics: It has distinctive diamond-shaped leaves with a white, mealy coating on the undersides. Common lambsquarters can produce numerous seeds and quickly establish in cultivated areas [33].

2.3.7. Kochia:

- **Behavior**: Kochia (Bassia scoparia) is known for its tolerance to drought and saline soils. It can spread rapidly, particularly in arid regions.

Characteristics: It has a bushy appearance with small, green leaves and can grow up to 6 feet tall. Kochia's adaptability and prolific seed production make it a challenging weed to control [34].

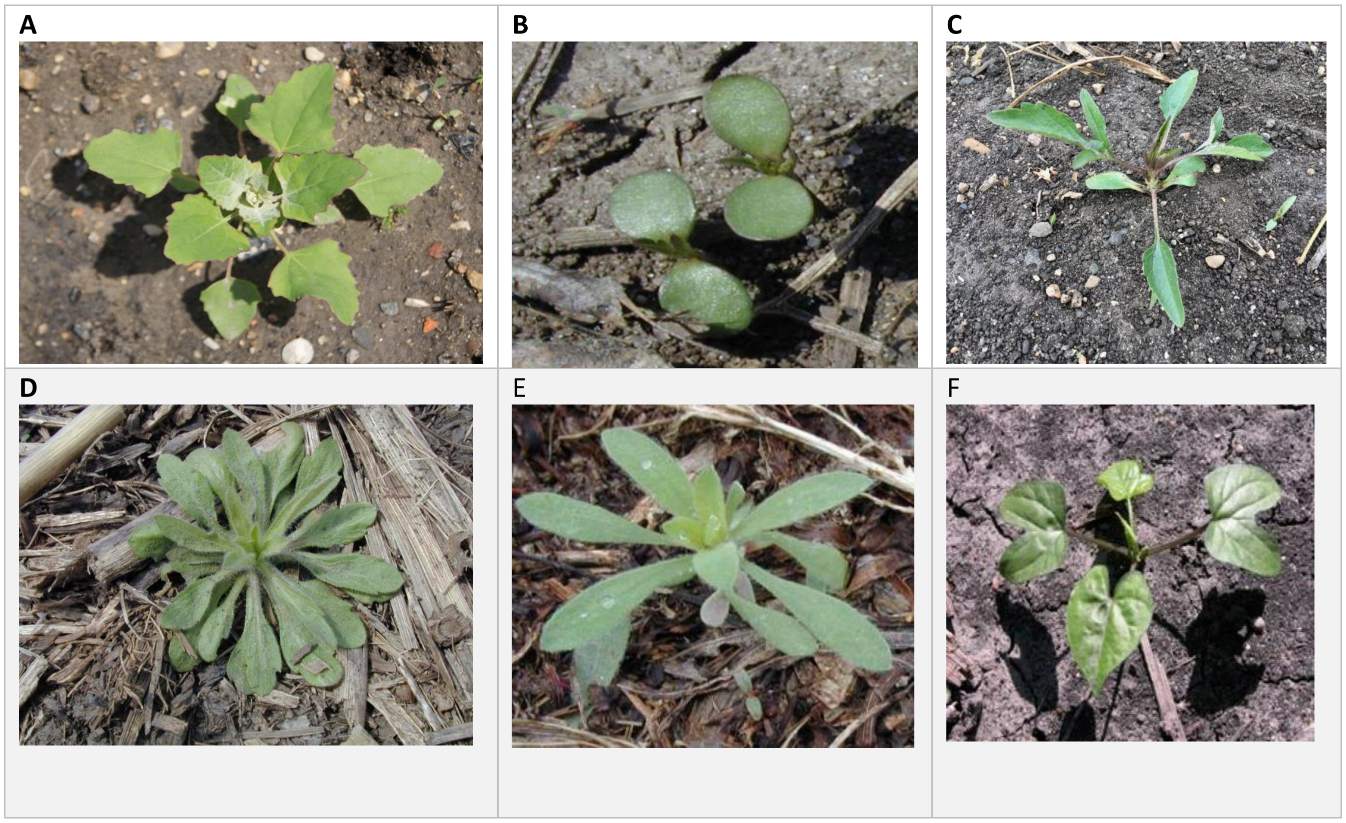


Fig 2- 2 A) Common Lambsquarters B) Common ragweed C) first true leaves of giant ragweed D) Horseweed (marestail) E) Kochia F) Morningglories [34]

2.4. Digital image processing

Digital image processing is a branch of computer science that deals with manipulating images. It is a broad field with numerous applications such as medical imaging, computer vision, and video surveillance.

Understanding the fundamentals of picture generation is helpful in understanding digital image processing. When light from an object meets a sensor, such as film in a camera or a digital sensor, a picture is created. The sensor turns light into electrical signals, which are subsequently recorded as an image in digital form in a computer.

A digital image is a two-dimensional array of numbers, where each number represents the brightness of a pixel at a particular location in the image. The number of bits used to represent each pixel is called the bit depth. A 24-bit image, for example, uses 24 bits to represent each pixel, which allows for a total of 16,777,216 possible colors. Figure1shows that how images are preprocess.

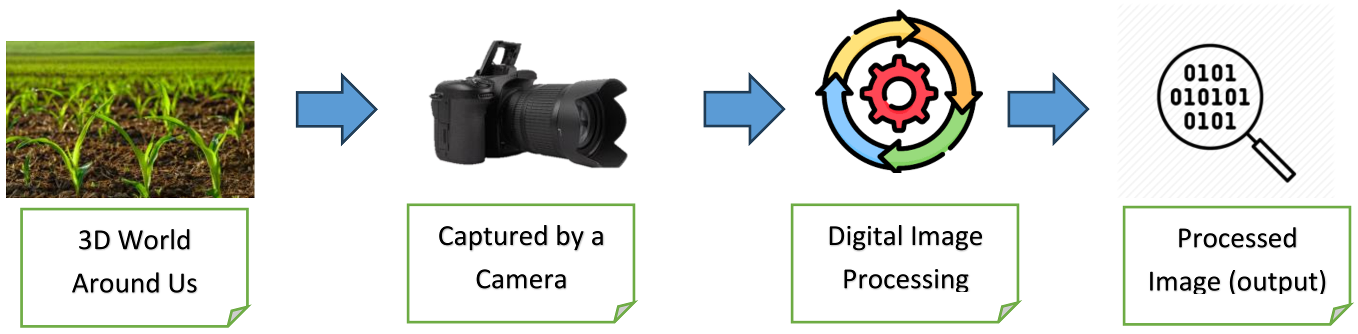


Fig 2- 3 How Digital Image processing works [35]

As noted by Rafael C. Gonzalez & Richard E [36]. Woods digital image processing techniques can be used to perform a wide variety of tasks on images, such as:

- Enhancement: This means improving an image's quality, such as by removing noise or changing the brightness and contrast.
- Segmentation: This involves dividing an image into different regions, such as by color or texture.
- Feature extraction: This involves extracting specific features from an image, such as edges or objects.
- Classification: This involves assigning labels to different regions in an image, such as by object type or scene type.
- Recognition: This involves identifying objects or scenes in an image.

Digital image processing is a powerful tool for improving image quality, analyzing, and understanding images. As noted, out by Rafael C. Gonzalez and Richard E [36]. Woods It has a wide range of applications, including:

- Medical imaging: Digital image processing is used to diagnose diseases, plan surgeries, and track the progress of treatments.
- Computer vision: Digital image processing is used to recognize objects, track motion, and understand the environment.

- Video surveillance: Digital image processing is used to monitor people and activities in public places.
- Remote sensing: Digital image processing is used to study the Earth's surface, atmosphere, and oceans.
- Entertainment: Digital image processing is used to create special effects in movies and video games.

Digital image processing is a fast-expanding discipline, with numerous new applications being created on a regular basis. As technology advances, digital image processing become an increasingly more powerful and adaptable tool.

Here are some additional benefits of digital image processing source:

- Accuracy: Digital image processing can be used to improve the accuracy of image analysis. For example, digital image processing can be used to remove noise from images, which can improve the accuracy of object detection and recognition.
- Speed: Digital image processing can be used to process images much faster than traditional analog methods. This can be useful for applications where speed is important, such as real-time video surveillance.
- Cost-effectiveness: Digital image processing can be a cost-effective way to improve the quality and accuracy of image analysis. This is because digital image processing software is becoming more affordable and powerful, and the cost of computing power is decreasing.

Overall, digital image processing is a powerful tool that can be used to improve the quality, analyze, and understand images. It is used in a wide variety of applications, and its benefits include accuracy, speed, and cost-effectiveness.

2.5. Steps in Digital Image Processing

Digital image processing contains a number of stages that must be completed in order to edit and analyze digital images. Each stage is critical in getting relevant information. The steps are as follows [36,37,38]:

Image acquisition: The initial stage in digital image processing is to acquire or obtain digital images using devices such as cameras or scanners. Real-world scenes are turned into digital image representations at this step. Scanners digitize actual prints or documents, whereas cameras acquire images by sensing light through sensors. To construct a discrete representation of the image, the analogue signals from the sensors are sampled and quantized.

Image acquisition is affected by a variety of elements such as sensor qualities, lighting conditions, and camera settings. To capture accurate and high-quality photos, it is critical to consider aspects like as resolution, color depth, and sensor sensitivity. The camera or scanner used has an impact on the total picture acquisition process.

Understanding the principles behind sensors, sampling, and quantization is critical for obtaining accurate and reliable digital images in image capture. This stage lays the groundwork for later processing steps like pre-processing, augmentation, and analysis.

Pre-processing: is an important step in digital image processing that seeks to improve image quality and prepare obtained images for analysis afterwards. It includes procedures like noise reduction, image scaling, and contrast modification.

Dealing with noise, which can decrease image quality and affect subsequent analysis, is one of the most difficult tasks in image processing. Spatial filtering and frequency domain filtering, for example, are used to suppress or eliminate undesirable noise while maintaining significant image elements. These strategies aid in improving image clarity and the accuracy of following processing steps.

Image resizing is another part of pre-processing that involves modifying the spatial dimensions of an image. Images can be resized to match certain display or analysis requirements. Working with reduced image sizes can also help minimize computing complexity.

Another significant operation in pre-processing is contrast adjustment, which tries to improve an image's dynamic range. Contrast enhancement techniques can bring out details that would otherwise be hidden or improve the visual appeal of an image by altering the pixel intensities.

Pre-processing is critical for ensuring that future image processing operations are performed on high-quality and well-prepared images. It reduces noise, resizes photos as appropriate, and adjusts contrast to improve overall image quality.

Image enhancement: is a crucial stage in digital image processing that focuses on improving an image's visual look. It seeks to improve overall perception by improving image quality, highlighting vital elements, and highlighting important information. Image enhancing techniques include histogram equalization and spatial filtering.

Histogram equalization is a technique that redistributes an image's pixel intensities to improve contrast. It expands the image's histogram to use the entire range of intensity values, resulting in increased visibility of details and overall image quality.

Another approach often employed in image enhancement is spatial filtering. It involves employing filters to change pixel values based on their spatial relationships in the image. Spatial filters can be used to do smoothing, sharpening, and edge detection. These filters help in the enhancement of image features and the overall visual appearance.

Histogram quantization and spatial filtering are both effective picture enhancement methods that allow image properties to be adjusted to produce desired visual enhancements. Images can be made visually appealing, more clearly and easier to interpret by employing these strategies.

Image segmentation: is a fundamental step in digital image processing that involves dividing an image into meaningful regions or objects based on their characteristics. The purpose is to divide the image into homogenous sections with similar attributes like color, intensity, texture, or motion.

One commonly used technique in image segmentation is thresholding. Thresholding divides the image into two or more regions based on a threshold value. Pixels with intensities above or below the threshold are assigned to different regions. This technique is effective for images with well-defined intensity distributions.

Region growing is another technique used for image segmentation. It starts from seed points or regions and iteratively expands these regions by adding neighboring pixels that meet certain similarity criteria. Region growing considers local pixel properties, such as intensity or color similarity, to determine the boundaries of the regions.

Image segmentation plays a vital role in various image analysis tasks, such as object recognition, tracking, and image understanding

Feature extraction is an important stage in digital image processing that involves finding useful image elements for future analysis. These features define distinct image qualities such as texture, form, or color and have significance in image processing tasks such as object recognition, image classification, and image retrieval.

Texture characteristics capture the spatial arrangement and patterns of pixels in a picture. They depict differences in intensity or color distribution, providing information about the textural features of different areas. Texture feature extraction techniques commonly used include Local Binary Patterns (LBP), Gray-Level Co-occurrence Matrices (GLCM), and Gabor filters. These methods capture different features of texture and can be used to distinguish between different textures in an image.

The geometric characteristics of the objects in an image are the focus of shape features. They talk about the shape's form, size, direction, and other characteristics. Techniques like boundary tracing, moment invariants, or Fourier descriptors can be used to extract shape characteristics. The tasks of object recognition, shape-based image retrieval, or character recognition can all benefit from these features.

The color distribution or characteristics of a picture are described by color features. They can be used to identify items or areas based on their color look and provide information about the color content. Techniques for obtaining color features include using color moments, color coherence vectors, or color histograms

In order to reduce the dimensionality of image data and obtain valuable information, feature extraction is essential. Following analytical tasks like image classification, object recognition, or content-based image retrieval use these derived features as inputs.

Image classification and recognition is a crucial stage in digital image processing that entails labelling or object identification based on the picture's retrieved attributes. It is essential to many applications, such as visual search, visual learning, and object detection.

In order to classify an image, a machine learning model is typically trained on a labelled dataset using the extracted features as input and the matching labels as output. The relationship between the features and the relevant object classes forms patterns and associations that the model learns to recognize. By predicting the most likely class based on the retrieved features, the model can categories new, unseen images once it has been trained.

2.6 Deep learning

Due to its capacity to handle complicated issues, the machine learning branch of deep learning has received a great deal of attention lately. Deep neural networks, in particular deep neural networks, which are modelled after the structure of the human brain, are at the foundation of deep learning. These networks are made up of several interconnected layers of nodes (neurons) that process and gain knowledge from data [39].

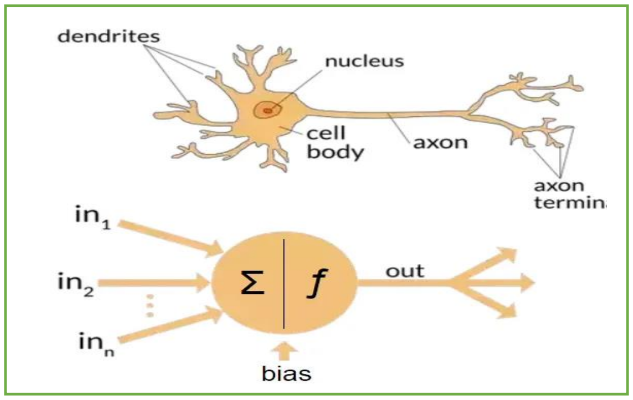


Fig 2- 4 A biological and an artificial neuron [39]

Over the years, deep learning has been crucial in enhancing digital image processing, resulting in appreciable gains in overall performance. With the help of this technology, processes like picture analysis, categorization, and recognition are now more precise and effective. Deep learning models have developed continuously to extract complex information from images with the introduction of Convolutional Neural Networks (CNNs) and deep architectures, leading to improved image quality and accuracy. The deep learning community has experienced a boom in research and innovation, as evidenced by the rise in publications and studies aimed at advancing deep neural network-based methods for digital image processing [39].

Deep learning is particularly effective at problems involving big datasets and high-dimensional data, such as speech recognition, image recognition, and natural language processing. It is appropriate for applications where human feature engineering is tough since it can automatically extract pertinent features from data [39].

2.7. Artificial Neural Network (ANN)

Neural networks are computational models for machine learning that are inspired by the structure of the biological brain. Neural networks are trained from examples rather than being explicitly programmed. Even with limited examples, neural networks can generalize and successfully deal with unseen examples [41].

ANN are machines designed to perform specific tasks by imitating how the human brain works, and build a neural network made up of hundreds or even thousands of artificial neurons or processing units. The artificial neural network is implemented by developing a computational learning algorithm that does not need to program all the rules since it is able to build up its own rules of behavior through what we usually refer to as “experience.”

An ANN at its smallest level is called the Perceptron. A perceptron is also known as a Threshold Logic Unit (TLU). It consists of a single unit also known as the neuron with multiple inputs and a single output. It is one of the simplest ANN architectures [42].

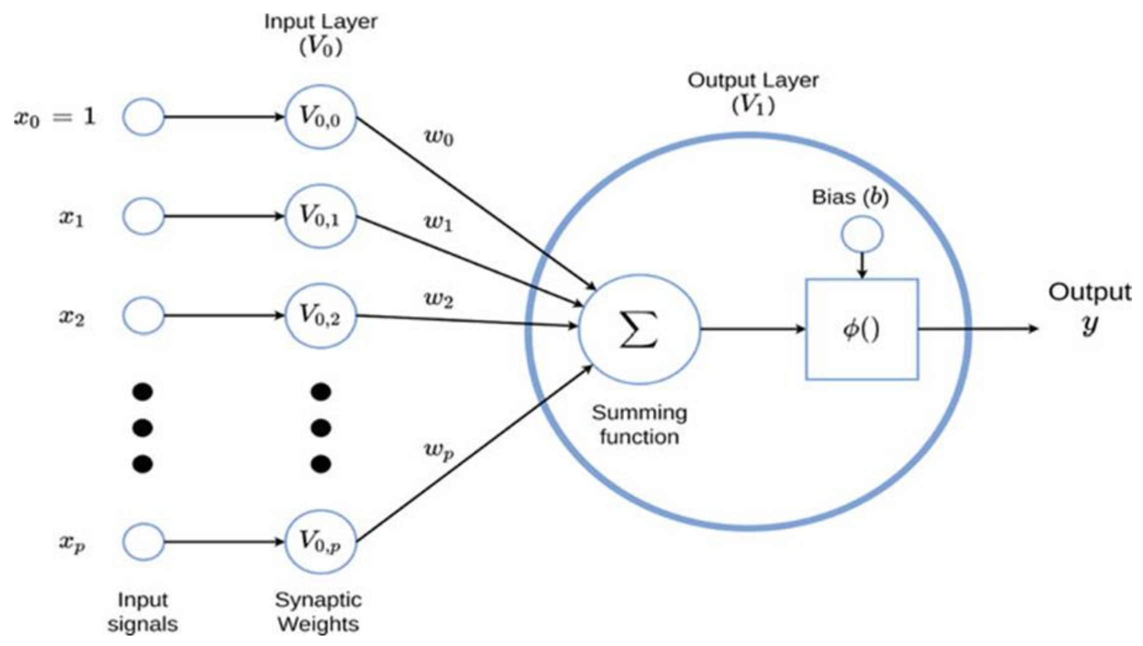


Fig 2- 5 General artificial neural network model [43]

An ANN at its smallest level is called the Perceptron. A perceptron is also known as a Threshold Logic Unit (TLU). It consists of a single unit also known as the neuron with multiple inputs and a single output. It is one of the simplest ANN architectures [41].

As depicted in figure 2-5 above, x1, . . ., xp represents the information (input) that the neuron receives from the external sensory system or from other neurons with which it has a connection. w = (w1, . . ., wp) is the vector of synaptic weights that modifies the received information emulating the synapse between the biological neurons. These can be interpreted as gains that can attenuate or amplify the values that they wish to propagate toward the neuron. Parameter 'bj' is known as the bias (intercept or threshold) of a neuron. Here in ANN, learning refers to the method of modifying the weights of connections between the nodes (neurons) of a specified network [43].

The different values that the neuron receives are modified by the synaptic weights, which then are added together to produce what is called the net input. In mathematical notation, that is equal to [43].:

$$v_j = \sum_{j=1}^{p} \omega_{ij} x_j$$

This net input (Vj) is what determines whether the neuron is activated or not. The activation of the neuron depends on what we call the activation function. The net input is evaluated in this function, and we obtain the output of the network as shown the previous fig [43].

the signals are passed between neurons through connection links. Each connection link has an associated weight, which, in a typical neuronal network, multiplies the transmitted signal. Each neuron applies an activation function (usually nonlinear) to the network inputs (sum of the heavy input signals) for determining its corresponding sign [43].

2.7.1. Activation Functions

The mapping between inputs and a hidden layer is determined by activation functions. Activation functions propagate the output of one layer's nodes forward to the next layer (up to and including the output layer). The activation function of a neuron (node) defines the functional form for how a neuron gets activated [43].

The activation function determines which value is high or low enough to reflect a decision point in the neural network for a particular neuron or group of neurons. As with everything else in neural networks, we don't have just one activation function. we use the activation function that works best in a particular scenario [44].

Activation functions are the basic building blocks of artificial neural networks. It determines the output of the neuron according to the weight of the inputs. [45].

2.7.2. Convolutional Neural Network (CNN or ConvNet)

CNN is a type of deep learning model for processing data that has a grid pattern, such as images which designed to automatically and adaptively learn spatial hierarchies of features, from low- to high-level patterns. A convolution layer plays a key role in CNN, which is composed of a stack of mathematical operations, such as convolution, a specialized type of linear operation. In digital images, pixel values are stored in a two-dimensional (2D) grid, i.e., an array of numbers and a small grid of parameters called kernel, an optimizable feature extractor, is applied at each image position, which makes CNNs highly efficient for image processing, since a feature may occur anywhere in the image [45].

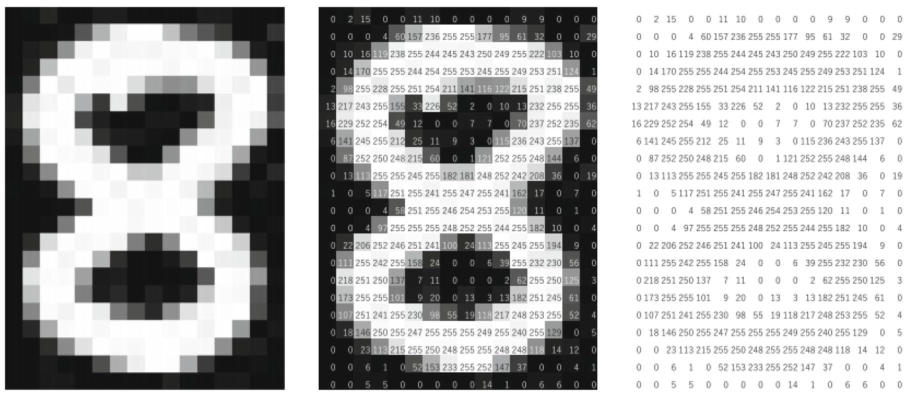


Fig 2- 6 In digital images, pixel values are stored in a two-dimensional (2D) grid [41]

CNN is a mathematical construct that is typically composed of three types of layers (or building blocks): convolution, pooling, and fully connected layers. The first two, convolution and pooling layers, perform feature extraction, whereas the third, a fully connected layer, maps the extracted features into final output, such as classification as shown in next fig (2-8).

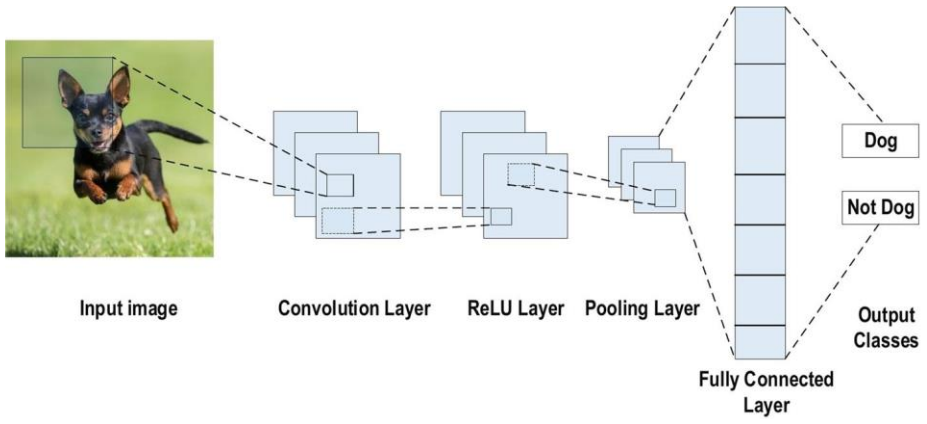


Fig 2- 7 An example of CNN architecture for image classification [46]

2.7.3. Layers of CNN (building blocks)

The CNN architecture consists of a number of layers (or so-called multi-building blocks). Each layer in the CNN architecture, including its function, is described in detail below [46].

A. Convolutional Layer

In CNN architecture, the most significant component is the convolutional layer. It consists of a collection of convolutional filters (so-called kernels). Each element of the kernel and the input tensor is calculated at each location of the tensor and summed to obtain the output value in the corresponding position of the output tensor, called feature map. This procedure is repeated applying multiple kernels to form an arbitrary number of feature maps, which represent different characteristics of the input tensors; different kernels can, thus, be considered as different feature extractor as shown in (**fig 2-9**) [45,46].

Kernel is grid of discrete numbers or values describes the kernel. Each value is called the kernel weight. Random numbers are assigned to act as the weights of the kernel at the beginning of the CNN training process. In addition, there are several different methods used to initialize the weights. Next, these weights are adjusted at each training era; thus, the kernel learns to extract significant features [41,42].

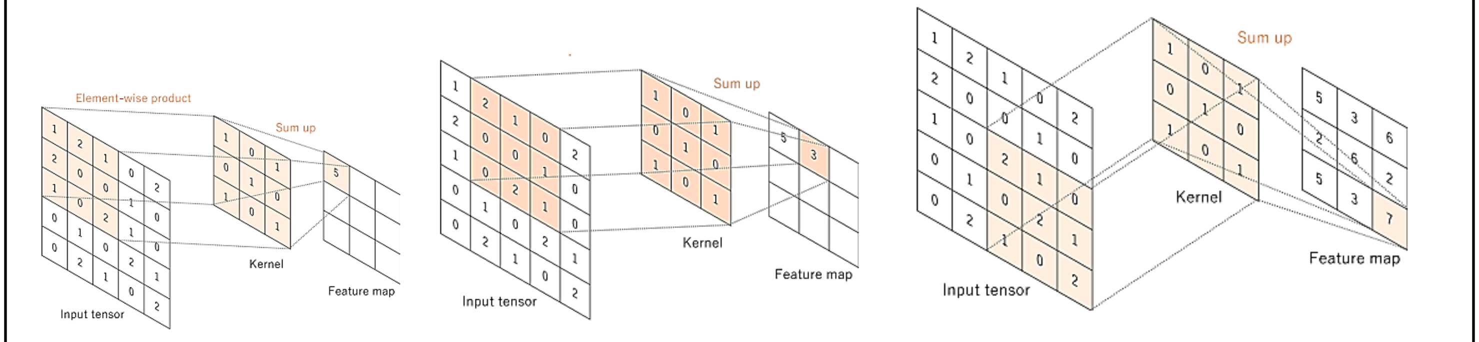


Fig 2- 8. An example of convolution operation with a kernel size of 3 ×3 [46]

Overall, the convolutional layer acts as a feature extractor by convolving learnable filters with input images, producing feature maps that represent different characteristics of the image. These extracted features serve as valuable inputs for subsequent layers in CNNs, enabling the network to learn complex patterns and make accurate predictions in tasks like image classification, object detection, and image segmentation [46].

B. Pooling layer

Pooling Layer: The main task of the pooling layer is the sub-sampling of the feature maps. These maps are generated by following the convolutional operations. In other words, this approach shrinks large-size feature maps to create smaller feature maps. Concurrently, it maintains the majority of the dominant information (or features) in every step of the pooling stage. In a similar manner to the convolutional operation, both the stride and the kernel are initially size-assigned before the pooling operation is executed. Several types of pooling methods are available for utilization in various pooling layers. These methods include tree pooling, gated pooling, average pooling, min pooling, max pooling, global average pooling (GAP), and global max pooling [45,46].

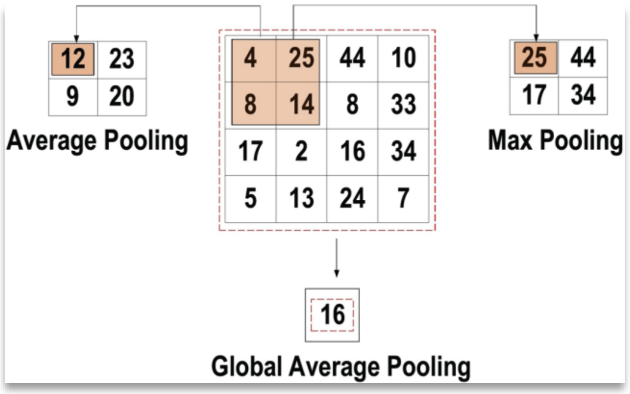


Fig 2- 9 Three types of pooling operations [42]

C. Fully Connected Layer

Commonly, this layer is located at the end of each CNN architecture. Inside this layer, each neuron is connected to all neurons of the previous layer, the so-called Fully Connected (FC) approach. It is utilized as the CNN classifier. It follows the basic method of the conventional multiple-layer perceptron neural network, as it is a type of feed-forward ANN. The input of the FC layer comes from the last pooling or convolutional layer. Once the features extracted by the convolution layers and down sampled by the pooling layers are created, they are mapped by a subset of fully connected layers to the final outputs of the network, such as the probabilities for each class in classification tasks. The final fully connected layer typically has the same number of output nodes as the number of classes. Each fully connected layer is followed by a nonlinear function (Activation function) [45,46].

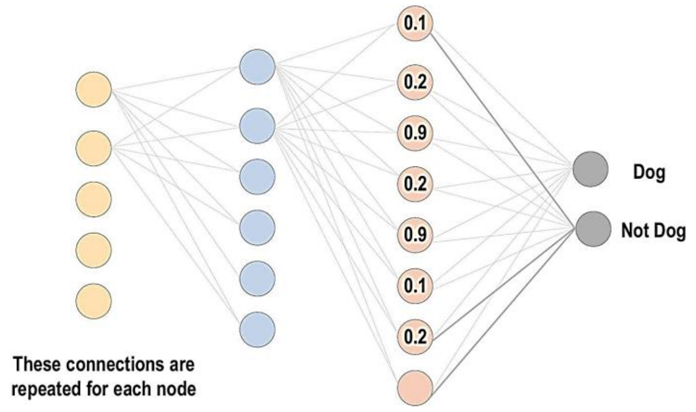


Fig 2- 10 Fully connected [46]

2.8. Related works

Weed detection in soybean crops using ConvNets is presented by Alessandro dos Santos Ferreira in [47]. The study gathered a vast image library encompassing over fifteen thousand photographs of soil, soybean, broadleaf, and grass weeds. They captured photographs of the soybean crops from an average altitude of 4 meters above ground level using a Phantom DJI 3 Professional drone. To ensure that the dataset was diverse and indicative of real-world situations, the photos were shot in different weather conditions and at different times of day.

The authors followed up by using the Simple Linear Iterative Clustering (SLIC) Super pixels technique to segment the images and help build an image dataset. The SLIC algorithm is a well-known image segmentation technique because of its efficiency and accuracy. It works by breaking an image into small, compact sections called super-pixels, from which features and machine learning models can be extracted.

The authors used a variety of feature extraction techniques some of them are: Gray-level co-occurrence matrix (GLCM): A feature extraction technique that maintains information about an image's texture. It computes the frequency of pairings of pixels with certain gray-level values and spatial correlations. The resulting matrix can be used to calculate characteristics like contrast, homogeneity, and entropy. they use in this work, in matrices 4 x 4 with distance 1 and 2 and angles 0°,45° and 90°, the following texture properties defined in the GLCM: energy, contrast, correlation, homogeneity and dissimilarity, resulting in 36 features. The authors of this work used GLCM to extract texture information from segmented images, also Local Binary Patterns (LBP) are considered.

Histogram of oriented gradients (HOG): HOG is a feature extraction technique that characterizes the shape and appearance of an object by the distribution of local intensity gradients or edge directions. It works by dividing an image into cells and calculating a gradient histogram for each cell. The resulting histograms can be concatenated to form a feature vector that describes the overall shape and appearance of the object. In this paper, the authors used HOG to extract shape features from the segmented images.

Local binary patterns (LBP): LBP is a feature extraction technique that is widely used for texture analysis. It works by comparing the intensity of a central pixel to the intensities of its neighboring pixels and encoding the result as a binary pattern. The resulting patterns can be used to extract features such as texture, contrast, and uniformity. In this paper, the authors used LBP to extract texture features from the segmented images.

For Training of the Convolutional Neural Networks (CNNs) they used the CaffeNet architecture, which is a replication of the well-known AlexNet network.

Overall, the authors trained and tested their ConvNet model using 400 soybean crop photos. The SLIC Super pixels algorithm was used to segment the dataset, and the segments were manually labelled with their appropriate classifications (soil, soybean, grass, and broadleaf weeds). The ConvNet model obtained 97.3% weed detection accuracy, which is much greater than the accuracy of other approaches proposed for this work.

Razfar, et al [48] presented a weed detection in soybean crops using custom lightweight deep learning models. As the research describe the paper proposes a vision-based weed detection system using custom lightweight deep learning models. The methodology involves collecting and labeling a dataset of images of soybean crops with and without weeds, training the deep learning models on this dataset, and evaluating their performance on a separate test dataset. The researchers also compare the performance of five different deep learning models in terms of accuracy and resource usage.

Collecting and labeling a dataset of images: The authors collected a dataset of 400 images of soybean crops with and without weeds. They manually labeled each image to indicate the presence or absence of weeds. They also segmented each image into 1536 total segments to increase the number of training examples.

Training deep learning models: The authors trained five different deep learning models on this dataset, including MobileNetV2, ResNet50, and three custom Convolutional Neural Network (CNN) models. They used transfer learning to fine-tune the pre-trained models on the soybean dataset. Transfer learning is a technique used to apply pre-trained models to new datasets. The authors used transfer learning to fine-tune the pre-trained MobileNetV2 and ResNet50 models on their soybean dataset. This involved retraining the models on the new dataset while keeping the pre-trained weights fixed. They trained three custom Convolutional Neural Network (CNN) models from scratch. Throughout the training process to optimize they used two techniques: dropout and model quantization.

Dropout is a regularization technique that helps prevent overfitting by randomly "turning off" activations during training. Since they are training many models, they might face an overfitting so the researchers placed a dropout layer in the single hidden layer of the MobileNetV2 model they proposed. However, they noted that this may not benefit much from the optimization since the model already has a small number of weights. the second one is model quantization which is a technique used to reduce the size and computation time of deep learning models. During training, the model represents each weight and activation using 32-bit floating-point numbers. Model quantization involves converting these data types to a smaller data type, such as a 16-bit floating-point or 8-bit integer, to reduce computation time and memory usage. The authors noted that this can significantly reduce computation time but may result in a loss of accuracy.

Overall, the authors evaluated the performance of each model on a separate test dataset of 100 images. They measured accuracy, latency, and memory usage for each model they compared the performance of the five models in terms of accuracy and resource usage. They found that the custom 5-layer CNN architecture showed the highest detection accuracy of 97.7% and the lowest latency and memory usage. This research main contribution is the development of a custom lightweight deep learning model that achieves state-of-the-art accuracy for weed detection in soybean crops. The model is also efficient and can be deployed on a low-cost microcontroller, making it suitable for real-time weed detection applications.

The proposed research in [49] involves using a deep learning approach to perform weed/crop classification in soybean crops. The authors propose a deep residual convolutional neural network (DRCNN) with contrast limited adaptive histogram equalization (CLAHE) for image enhancement.

The DRCNN architecture is designed to improve gradient flow through the network, which can help to avoid the vanishing gradient problem that can occur in deep neural networks. The residual connections in the network allow for the reuse of features learned in earlier layers, which can help to improve the accuracy of the network.

The main branch of the DRCNN architecture is divided into five sections, as described in on Page of the PDF file:

(i) Image input layer, (ii) Three stages of convolutional layers with different feature sizes: 32x32, 16x16, and 8x8, respectively, with each stage containing 2 convolutional units, (iii). The last section contains global average pooling, fully connected layer, softmax layer, and classification layer.

The main branch's convolutional layers are designed to extract features from the input images, while the shortcut connections allow the network to reuse features learnt in previous levels. The fully connected layer is used to map the features to the output classes, while the global average pooling layer is utilized to minimize the spatial dimensions of the feature mappings. The softmax layer converts the fully connected layer's output into a probability distribution over the output classes, and the classification layer makes the final prediction.

CLAHE is used to increase the contrast of images, which can help the network's accuracy. CLAHE is a technique that improves image contrast by splitting it into segments and performing histogram equalization to each zone individually. This can help in the visibility of small details in the image, which is useful for weed/crop classification. The CLAHE algorithm can be summarized in the following steps:

I.) Divide the image into small tiles, (II) Compute the histogram of each tile, (III) Apply histogram equalization to each tile separately, (IV) Clip the pixel values in each tile to a maximum value to avoid over-amplification of noise, (V) Combine the enhanced tiles to form the final image.

CLAHE is used to enhance the contrast of the input images before they are fed into the deep residual convolutional neural network (DRCNN) for weed and crop classification. This can help to improve the accuracy of the DRCNN by making it easier to detect the features that distinguish weeds from crops.

Overall, the proposed methodology achieved a validation accuracy of 97.25%. This means that the model was able to correctly classify 97.25% of the validation images, which is a measure of how well the model can generalize to new, unseen data.

Table 2- 2 Summary of the related works

No	Author (year)	Statement of the problem	Approach's	Result	Research gap
1	Alessandro, et al. (2021) **[47]**	To develop an effective and efficient method for detecting and classifying weeds in soybean crops using CNN techniques.	segmenting images using super-pixels, and convolutional Nural Networks (CNNs) to classify the segmented images	CNNs achieved high accuracy with an average accuracy of 99.5 %	lack of a large and diverse (Weed Species Diversity)
2	Razfar, et al. (2022) [48]	Weeds are responsible for 45% of crop losses in the agriculture industry, mainly due to competition with crops.	Use CNN models, like MobileNetV2, ResNet50, and three custom CNN models	The custom 5-layer CNN architecture achieved accuracy of 97.7% on the validation set. The MobileNetV2 and ResNet50 achieved validation accuracies of 96.5% and 96.2%,	relatively small, with only 1,000 images. The authors note that a larger dataset could improve the performance of the models.
3	Babu1, et al. (2022) [49]	Overuse of herbicides in farming can have negative impacts on the environment and human health. Therefore, there is a need for a more sustainable and efficient approach to weed management in agriculture.	Deep learning techniques, specifically a deep residual convolutional neural network (DRCNN) with contrast limited adaptive histogram equalization (CLAHE), to accurately detect and classify weeds in soybean crops.	achieved high accuracy with an average accuracy of 97.25%	The authors only used one dataset consisting of images of soybean crops and weeds, which may limit the generalizability of the proposed approach to other crops and weed species

The presented review of related works in weed detection using deep learning techniques provides insights into the advancements made by various researchers in the agricultural domain. However, it also reveals certain research gaps that the current study aims to address.

Limited Exploration of Convolutional Neural Networks (CNNs) in Weed Detection:

The first related work by Ferreira et al. [47] focuses on weed detection in soybean crops using ConvNets. While the study demonstrates the effectiveness of ConvNets in achieving high weed detection accuracy (97.3%), it primarily explores traditional feature extraction techniques such as Gray-level co-occurrence matrix (GLCM), Histogram of Oriented Gradients (HOG), and Local Binary Patterns (LBP) alongside ConvNets. The research gap lies in the limited exploration of the potential of deep learning models beyond ConvNets and their combinations with traditional techniques for improved weed detection.

Inadequate Exploration of Image Enhancement Techniques:

The third related work [49] proposes a deep residual convolutional neural network (DRCNN) with contrast limited adaptive histogram equalization (CLAHE) for weed/crop classification in soybean crops. While the study successfully achieves a high validation accuracy of 97.25%, there is a research gap in the limited exploration of various image enhancement techniques. The study primarily focuses on CLAHE, and there is room for investigating the impact of other image preprocessing methods on the model's performance, considering that different techniques may have varied effects on contrast improvement and feature extraction.

In conclusion, the identified research gaps emphasize the need for the current study to explore novel deep learning architectures, diverse image enhancement techniques, and robust generalization to address the evolving challenges in weed detection in soybean crops. The study aims to contribute to the existing body of knowledge by filling these gaps and advancing the understanding of deep learning applications in precision agriculture.

CHAPTER THREE

Methods

In this pivotal chapter, a comprehensive analysis of the experimental results is done towards training the custom convolutional neural network (CNN) model. The resounding success of achieving a peak test accuracy of 94.67% underscores the efficacy of the model in recognizing and classifying images across multiple classes. This chapter provides a detailed narrative of the experimental journey, offering insights into the choices made, challenges encountered, and the ultimate triumph of the custom-built CNN in tackling the multi-class image classification task.

3.1 Proposed Architecture

In this section, we present the generalized architecture underpinning the employed model for soybean and weed detection. The proposed architecture is based on a Convolutional Neural Network (CNN) framework, tailored to excel in image classification tasks. The design is versatile, adaptable to various datasets and tasks within the domain.

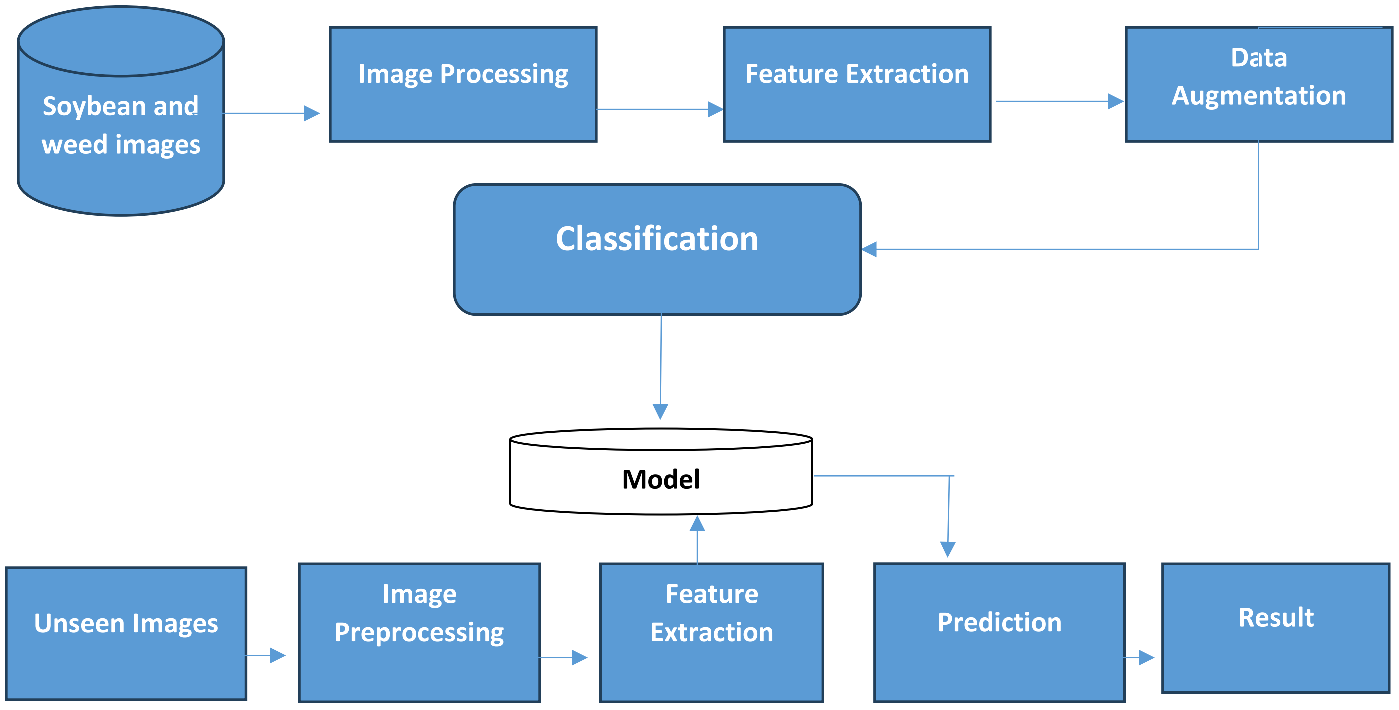


Fig 3- 1model Architecture

3.1.2. Overview of the Architecture

The architecture is designed to process images of varying dimensions, accommodating the diversity inherent in agricultural datasets. The initial layer serves as the input layer, capable of handling images with variable width, height, and color channels (RGB).

Subsequently, a series of convolutional layers are employed to extract features hierarchically. These layers are followed by max-pooling operations, contributing to the model's ability to discern spatial information. The number of filters in these convolutional layers increases progressively, allowing the model to capture intricate patterns.

Following the convolutional layers, a flatten layer transforms the output into a 1D vector, preparing it for subsequent fully connected layers. The architecture incorporates two dense layers with rectified linear unit (ReLU) activation. A dropout layer is strategically placed after each dense layer to mitigate overfitting, promoting better generalization.

The final layer, utilizing softmax activation, is suitable for multi-class classification tasks. The number of neurons in this layer corresponds to the classes in the dataset, ensuring adaptability to different classification scenarios.

3.1.3 Model Compilation and Optimization

The model is compiled using the Adam optimizer, offering efficiency and adaptability. Categorical crossentropy loss is employed for training, and accuracy serves as the monitoring metric. These choices contribute to the model's ability to converge effectively and provide meaningful predictions.

3.1.4. Data Augmentation Strategies

To enhance the model's generalization capabilities, various data augmentation techniques are incorporated, including rotation, shift, shear, zoom, and horizontal flip. These techniques contribute to the robustness of the model by exposing it to diverse perspectives of the input data.

3.1.5. Significance of the Architecture

This architecture is crafted to be versatile and applicable across a spectrum of image classification tasks within the context of soybean and weed detection. Its adaptability to different image dimensions, coupled with robust feature extraction and classification capabilities, positions it as a valuable tool for addressing the complexities inherent in agricultural datasets. The emphasis on spatial information, hierarchical feature extraction, and generalization through data augmentation collectively contribute to the efficacy of the proposed architecture.

3.2 Data Collection

3.2.1 Data Sources

Identification of the Primary Sources of Data

The primary sources of data for this study were Kaggle and Google. Kaggle provided publicly available datasets, while Google was accessed through a custom script capable of querying and downloading images based on specific search terms. The datasets covered a range of scenarios, capturing variations in crop density, weed species, and environmental conditions.

Detailed Explanation of the Datasets Used

The datasets included broadleaf (2114 images), grass (3520 images), soil (3249 images), and soybean (7376 images). Kaggle datasets contributed significantly, and the custom script filled gaps by fetching images from Google based on predefined queries. This diverse dataset composition was essential for training a model that could accurately classify soybean and weed instances in various agricultural contexts.

This script serves as a convenient tool for downloading images related to a specific query from Google. It encapsulates this functionality within a modular function (download_images) and provides a user-friendly interface through the main function. Users can easily set their search query, limit the number of images to download, specify the folder for storage, and run the script to automate the image retrieval process. It's important to note that the script relies on the ‘pygoogle_image’

3.2.2 Data Preprocessing

Description of Data Cleaning and Transformation Procedures

The data preprocessing stage focused on enhancing the quality and uniformity of the collected images. A background removal script was applied to certain datasets, particularly the soybean dataset. This script utilized color space transformations and edge detection techniques to isolate plant regions, effectively blacking out non-plant areas. This preprocessing step aimed to improve the model's focus on relevant features during training.

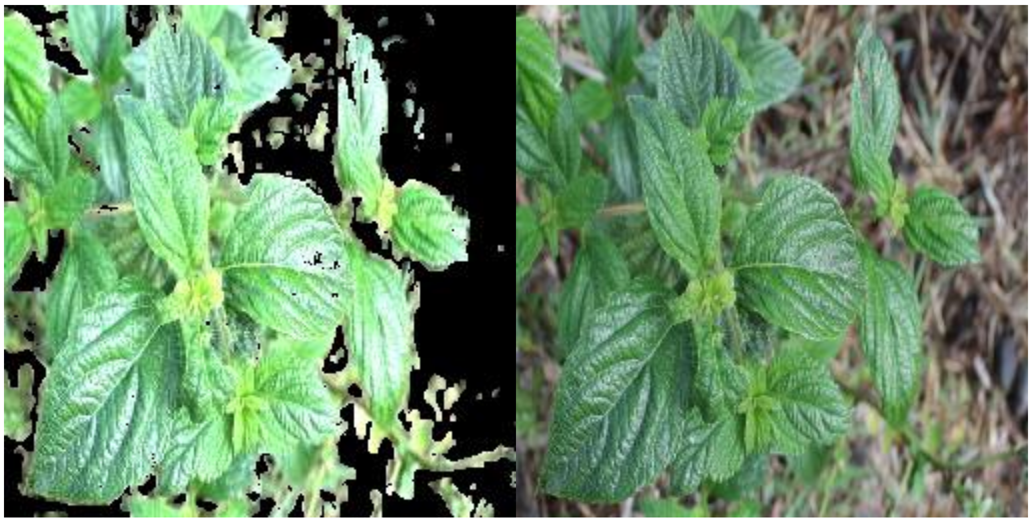


Fig 3- 2 Before and After Handling Missing or Noisy Data

3.2.3. Handling Missing or Noisy Data

Careful consideration was given to handling missing or noisy data during the preprocessing stage. The background removal script ensured that irrelevant background elements were eliminated, reducing noise in the dataset. Additionally, thorough quality checks were performed to address any anomalies in the acquired images, promoting a cleaner and more reliable dataset.

Input Parameters:	**Parameters:**	**Functionality:**
input_folder: The path to the folder containing input images.	Input_folder: The folder containing the original images.	Loops through each file in the specified input folder.
output_folder: The path to the folder where the processed images will be saved.	output folder: The folder where processed images will be saved.	Reads the input image and converts it to the HSV color space.
	Brightness_factor: A factor to control the brightness of the processed image.	Defines a color range for green in HSV and creates a mask for green areas.
	canny_threshold1 and canny_threshold2: Parameters for the Canny edge detector.	Applies Canny edge detection to the green mask to highlight edges.
	dilation_kernel_size: The size of the kernel for dilation operation.	Combines the green mask and edges mask using bitwise OR operation.
	blur_kernel_size: The size of the kernel for Gaussian blur.	Applies dilation to expand the detected regions.
		Sets non-green areas to black, effectively isolating green areas.
		Applies Gaussian blur to smooth the foreground.
		Increases the brightness of green areas.

Table 3- 1 Handling Missing or Noisy Data

3.2.4. Image Preprocessing

The image preprocessing pipeline in the provided code encompasses several crucial steps to prepare the dataset for effective training and evaluation within the Convolutional Neural Network (CNN) architecture. Initially, images are loaded using OpenCV and stored as NumPy arrays. Subsequently, to ensure uniformity in input dimensions, the images undergo resizing to a consistent size of 200x200 pixels, a crucial step for facilitating consistent model input. Augmentation techniques, including rotation, width shift, height shift, shear, zoom, and horizontal flip, are then applied using Keras' `ImageDataGenerator`. This intentional introduction of variations in the training dataset aids the model in generalizing well to diverse input scenarios. Finally, a critical normalization step is implemented, where pixel values of the images are scaled to the range [0, 1] by dividing each pixel value by 255.0. This normalization not only ensures faster convergence during training but also contributes to overall model performance. Collectively, these preprocessing steps lay a robust foundation for training a CNN to effectively learn and classify features from the input images.

3.3. Data Augmentation

Data augmentation is a vital technique in machine learning, especially for tasks like image classification. The primary motivation behind data augmentation is to address the challenge of limited training data. In many cases, having a small dataset may lead to overfitting, where the model becomes too specific to the training samples and struggles to generalize well to new, unseen data. Data augmentation aims to artificially expand the dataset by applying various transformations to the existing images, creating new, realistic variations. This process helps expose the model to a more diverse set of examples, enhancing its ability to learn robust features and improving generalization performance.

Data augmentation involves applying a set of predefined transformations to the training dataset, creating augmented versions of the original samples. These transformations simulate real-world variations that the model might encounter during deployment. Common augmentations include rotation, flipping, scaling, cropping, and changes in brightness or contrast. The goal is to make the model invariant to these variations, ensuring it performs well on a wide range of inputs.

Data augmentation, a pivotal strategy in the provided model, is implemented through the use of Keras' `ImageDataGenerator` class. This technique systematically introduces variations to the training dataset, artificially diversifying it and fortifying the model against overfitting while enhancing its ability to generalize to unseen scenarios. The `ImageDataGenerator` is configured to apply a range of transformations, including rotation by up to 40 degrees, random horizontal and vertical shifts, shear transformations for deformations, random zooming to simulate different distances, and the occasional horizontal flip. These transformations are crucial for exposing the neural network to a broader spectrum of visual patterns, orientations, and perspectives, making it more robust and adaptable to real-world scenarios. By seamlessly integrating data augmentation into the model training process, the code ensures that augmented batches are generated on-the-fly during each training epoch, contributing significantly to the model's overall effectiveness and performance.

3.4. Data Augmentation for Proposed Mode

rotation_range	*40*
width_shift_range	0.2
height_shift_range	0.2
shear_range	0.2
zoom_range	0.2
horizontal_flip	True
fill_mode	'nearest'
rotation_range	40

Table 3- 2 Augmentation techniques that have been used on proposed model

In the provided table, the following augmentation techniques are applied using the `ImageDataGenerator` class from Keras:

- ✓ **Rotation**: - Images are rotated by a maximum angle of 40 degrees. This introduces variations in the orientation of the images, allowing the model to be robust to different angles of view.
- ✓ **Width and Height Shift**: - Random horizontal and vertical shifts are applied to the images with a maximum shift of 20% of the image width and height. This helps the model learn from slightly translated versions of the images.

- ✓ **Shear Transformation**: - Random shear transformations are applied with a maximum shear intensity of 20 degrees. This introduces deformations in the images, enhancing the model's ability to recognize patterns under different shapes.
- ✓ **Zoom**: - Random zooming is applied to the images with a maximum zoom range of 20%. This simulates the effect of images taken from different distances and scales.
- ✓ **Horizontal Flip**: - Images are horizontally flipped with a probability of occurrence. This augmentation introduces variations in the orientation of objects, enhancing the model's ability to recognize features regardless of their orientation.
- ✓ **Fill Mode**: - The `fill_mode` parameter is set to 'nearest,' which determines the strategy used for filling in newly created pixels resulting from rotations or shifts. 'Nearest' filling mode is commonly used to maintain the integrity of the image content.

These augmentation techniques collectively contribute to the model's robustness by exposing it to diverse variations in the training data. The goal is to enable the model to generalize well to unseen data and real-world scenarios.

3.5. Classification

Classification in the context of digital image processing or Convolutional Neural Networks (CNNs) refers to the task of assigning a label or category to an input image based on its visual content. It is a fundamental problem in computer vision and is widely used in various applications such as object recognition, image segmentation, and medical image analysis [51].

3.5.1 Image Classification in Digital Image Processing: Traditional Classification Approaches

Image classification in digital image processing is a crucial task involving the assignment of labels or categories to images based on their visual content. Traditional classification approaches in this context often rely on a combination of feature extraction and machine learning techniques [50].

In the process of traditional image classification, the first step is to identify and extract relevant features from the images. These features could include color histograms, texture patterns, shape descriptors, or other distinctive characteristics that can capture the essence of the visual content in the images. Feature extraction plays a critical role in distinguishing between different classes of images [50].

Gonzalez and Woods (2017) emphasize the importance of utilizing machine learning classifiers for the actual classification task. One common approach is the use of Support Vector Machines (SVM), a powerful algorithm that aims to find the optimal hyperplane to separate different classes in the feature space. SVMs are particularly effective when dealing with high-dimensional data, making them suitable for image classification tasks where features extracted from images form a complex and multidimensional space [50].

Decision Trees represent another traditional classification technique employed in image processing. These trees consist of nodes representing decisions based on features and branches corresponding to possible outcomes or further decisions. Decision Trees are interpretable and can capture complex decision boundaries, making them valuable for image classification tasks where the relationships between features are non-linear [50].

While traditional approaches in digital image processing have been successful, they often require manual feature engineering, which can be a time-consuming and challenging process. The evolution of deep learning, especially Convolutional Neural Networks (CNNs), has shifted the paradigm by automating the feature learning process [50].

In conclusion, traditional classification approaches in digital image processing involve the extraction of meaningful features from images, followed by the application of machine learning classifiers such as Support Vector Machines or Decision Trees. The book "Digital Image Processing" by Gonzalez and Woods (2017) serves as a comprehensive reference for understanding the principles and techniques associated with image classification in the realm of digital image processing [50].

3.5.2 Image Classification in Convolutional Neural Networks (CNNs):

Image classification in Convolutional Neural Networks (CNNs) represents a significant advancement in the field of computer vision. CNNs are a class of deep neural networks specifically designed for image-related tasks, and they have demonstrated remarkable success in automatic feature learning and image classification [51].

CNNs have proven to be highly effective for image classification tasks. CNNs are a type of deep learning architecture specifically designed to work well with grid-like data, such as images. They automatically learn hierarchical features from the input images through a series of convolutional layers and pooling layers.

CNN Architecture and Operation:

The architecture of a CNN is inspired by the visual processing in the human brain. Convolutional layers, the core components of CNNs, automatically learn hierarchical features from the input images. These layers apply convolutional operations to detect local patterns and spatial relationships, enabling the network to capture complex visual representations. Activation functions like Rectified Linear Unit (ReLU) introduce non-linearity, enabling CNNs to learn more intricate patterns [51].

Training Process:

CNNs are trained through a process involving a loss function, backpropagation, and optimization. During training, the network learns to minimize the difference between predicted and true labels. Backpropagation is utilized to adjust the parameters of the network, optimizing its ability to make accurate predictions. Various optimization techniques, such as stochastic gradient descent (SGD) and adaptive methods like Adam, contribute to the efficiency of the training process [52].

Image Classification Process:

The image classification process in CNNs involves presenting an image as input, extracting features through convolutional layers, and producing class probabilities in the final layer. The softmax activation function is commonly used in the output layer to generate a probability distribution over different class. The class with the highest probability is then assigned as the predicted label [53].

Types of image classification

There are several types of image classification methods. The choice of method depends on the complexity of the problem, available data, and the specific requirements of the application [54].

Here are some common types of image classification [54]:

- **Binary Classification**: Binary classification involves categorizing images into two classes. For example, distinguishing between images of cats and dogs, or identifying whether an image contains a specific object or not.
- **Multi-Class Classification**: In multi-class classification, images are classified into more than two classes. Each image is assigned a label from a set of predefined categories. For instance, classifying images into categories like animals, vehicles, or buildings.
- **Fine-Grained Classification**: Fine-grained classification focuses on differentiating between subclasses within a broader category. It involves classifying images with subtle visual differences, such as distinguishing between different species of birds or different types of flowers.
- **Hierarchical Classification**: Hierarchical classification involves organizing classes into a hierarchical structure. Images are first classified at a broader level and then further classified into more specific subcategories. This approach enables a more organized and granular classification process.
- **Object Detection and Localization:** Object detection goes beyond simple classification by not only identifying the presence of objects but also localizing their positions within the image. It involves drawing bounding boxes around the objects of interest.
- **Semantic Segmentation:** Semantic segmentation assigns a class label to each pixel in an image, dividing it into meaningful regions. This technique enables detailed understanding of the image's content and is commonly used in applications like autonomous driving, medical imaging, and scene understanding.
- **Weakly Supervised Classification:** Weakly supervised classification is a technique where the training data is labeled at a coarse level, such as image-level labels, instead of pixel-level annotations or bounding boxes. The model learns to classify images using this limited supervision, making it more cost-effective and scalable for large-scale datasets.
- **Transfer Learning and Optimization**: Transfer learning is a prevalent technique in CNNs where pre-trained models on large datasets are fine-tuned for specific tasks with smaller datasets. This approach leverages the knowledge gained from general image recognition tasks and adapts it to the specifics of the new classification task. Additionally, optimization techniques play a crucial role in enhancing the performance of CNNs, ensuring they generalize well to diverse datasets [55].

3.6. Data splitting after augmented images for Proposed Model

In the provided model, the data splitting is done after the application of data augmentation. The dataset is split into training, validation, and test sets after augmenting the images. Let's break down the process:

- ✓ **Data Augmentation**: - Before splitting the dataset, the `ImageDataGenerator` is used to generate augmented batches on-the-fly during model training. This involves applying various transformations such as rotation, width shift, height shift, shear, zoom, and horizontal flip to the original images.
- ✓ **Training Data Flow**: - The augmented batches are then used to train the neural network. The `datagen.flow` method is used to generate these batches on-the-fly, and the augmented images are used as input to the model during training.
- ✓ **Data Splitting**: - After completing the training with augmented data, the dataset is split into training, validation, and test sets using the `train_test_split` function from scikit-learn. This is done to evaluate the model's performance on data it has not seen during training.
- ✓ **Normalization**: - After splitting, the images are normalized to the range [0, 1] before feeding them into the neural network.

  This approach ensures that the data splitting is performed on the augmented dataset, allowing the model to be evaluated on both original and augmented samples during the validation and test phases.

3.7. Training Hyperparameters:

The hyperparameters used in the proposed model, include various settings for the neural network architecture, data augmentation, and training. Here's a summary of the key hyperparameters:

Neural Network Architecture:

Convolutional Layers	Pooling Layers	Flatten Layer	Fully Connected (Dense) Layers	Output Layer
- Four convolutional layers are used with1 6, 32, 64, 128, and 256 filters, respectively.	Max pooling layers follow each convolutional layer with a (2, 2) pool size.	A flatten layer is employed to transform the 3D output into a 1D vector.	Two fully connected layers with 512 and 256 neurons, respectively, both using ReLU activation.	The output layer has neurons equal to the number of classes (4 in this case) with softmax activation for multi-class classification
Each convolutional layer uses a (3, 3) kernel size and the rectified linear unit (ReLU) activation function.			Dropout layers with a dropout rate of 0.5 are added after each dense layer to reduce overfitting.	

Table 3- 3 Hyper-parameters that have been used on proposed model

These hyperparameters collectively define the architecture of the neural network, the optimization strategy, and the parameters for data augmentation during training. Adjusting these hyperparameters can have a significant impact on the model's performance, and finding the optimal values often involves experimentation and fine-tuning.

3.8. Image Feature Extraction for Proposed Model

The primary technique for feature extraction is convolution. Convolutional layers apply filters to input images, capturing low-level features in early layers and high-level, abstract features in deeper layers. Max pooling is employed to down-sample the spatial dimensions and focus on the most relevant information. The combination of these techniques allows the model to automatically learn and extract hierarchical features, enabling effective image classification.

In the provided study, the image feature extraction is performed by the convolutional layers of the Convolutional Neural Network (CNN). The convolutional layers are designed to automatically learn and extract hierarchical features from the input images. Here is how feature extraction is achieved in the proposed model:

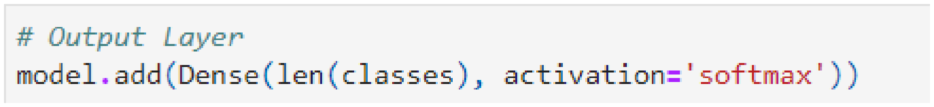
- ✓ **Convolutional Layers**: The core of the feature extraction process is the stack of convolutional layers. These layers learn filters or kernels that automatically detect patterns, textures, and features in the input images.
- ✓ Each convolutional layer convolves the input image with a set of filters to produce feature maps. These feature maps represent the presence of specific features or patterns in different regions of the image. The number of filters increases in deeper layers, allowing the network to learn increasingly complex and abstract features.
- ✓ **Max Pooling Layers**: Max pooling layers follow each convolutional layer, reducing the spatial dimensions of the feature maps and focusing on the most important information. Pooling helps achieve translation invariance, making the model robust to slight changes in object position.
- ✓ **Flatten Layer**: After the convolutional layers, a flatten layer is employed to transform the 3D feature maps into a 1D vector. This flattening step prepares the features for input to the fully connected layers.
- ✓ **Fully Connected (Dense) Layers**: The flattened features are then passed through fully connected (dense) layers, which further process and combine the learned features. These layers have 512 and 256 neurons, respectively, with ReLU activation. The use of ReLU activation functions introduces non-linearity, enhancing the model's ability to learn complex patterns. Additionally, dropout is applied to prevent overfitting during training, improving the generalization capability of the model.

The convolutional layers, in essence, act as feature extractors by learning filters that highlight relevant patterns in the images. The subsequent layers further refine these features, ultimately leading to a representation that can be used for classification. It's important to note that the final layer, with softmax activation, is responsible for generating the model's output probabilities for each class, effectively making the model capable of classifying images based on the learned features.

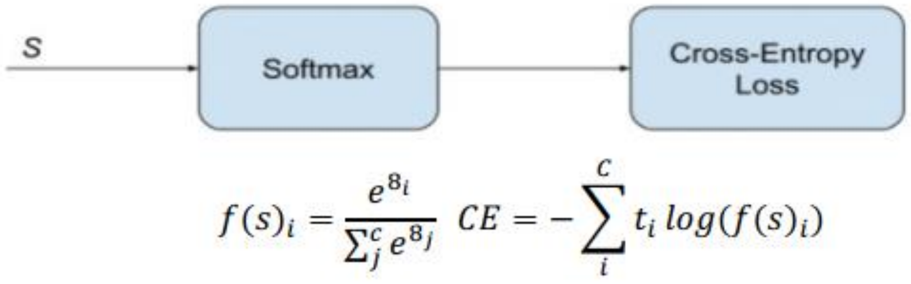
3.9. Loss and Metrics for the Model

In the provided code, the model is compiled with the following loss function and evaluation metric:

✓ **Loss Function:** The loss function used for training the model is categorical Cross-entropy (SoftMax). This is a common choice for multi-class classification problems where each input sample can belong to only one class. The categorical cross-entropy loss is suitable for optimizing models that output a probability distribution over multiple class.



```
# Output Layer
model.add(Dense(len(classes), activation='softmax'))
```



$$f(s)_i = \frac{e^{8_i}}{\sum_j^c e^{8_j}} \quad CE = -\sum_i^C t_i \log(f(s)_i)$$

Fig 3- 3 . Loss of Categorical Cross-Entropy [56]

In the specific case of multi-Class classification, the labels are one hot, so only the positive class ***Cp*** keeps its term in the loss. There is only one element of the Target vector ***t*** which is not zero ***ti*** = ***tp***. So, discarding the elements of the summation, which are zero due to target labels, we

can write:

$$CE = -log\left(\frac{e^{s_p}}{\sum_j^c e^{s_j}}\right)$$

Where ***Sp*** is the CNN score for the positive class.

After defining the loss, we must compute its gradient in relation to the CNN's output neurons in order to back propagate it through the net and optimize the defined loss function by modifying the net parameters. As a result, we must compute the Cross Entropy Loss gradient for each CNN class score in **s**. The loss terms from the negative classes are all equal to zero. However, because the Softmax of the positive class is equally dependent on the scores of the negative classes, the loss gradient with regard to those negative classes is not cancelled.

The gradient expression will be the same for all ***C*** except for the ground truth class ***Cp***, because the score of ***Cp*** (***sp***) is in the nominator.

After some calculus, the derivative respect to the positive class is:

$$\frac{\partial}{\partial_{s_p}}\left(-log\left(\frac{e^{s_p}}{\sum_j^c e^{s_j}}\right)\right) = \left(\frac{e^{s_p}}{\sum_j^c e^{s_j}} - 1\right)$$

In addition, the derivative respect to the other (negative) classes is:

$$\frac{\partial}{\partial_{s_n}}\left(-log\left(\frac{e^{s_p}}{\sum_j^c e^{s_j}}\right)\right) = \left(\frac{e^{s_n}}{\sum_j^c e^{s_j}}\right)$$

Where ***sn*** is the score of any negative class in *C* different from ***Cp***.

When Softmax loss is used is a multi-label scenario, the gradients get a bit more complex, since the loss contains an element for each positive class. Consider ***M*** are the positive classes of a sample. The CE Loss with Softmax activations would be:

$$CE = \frac{1}{M}\sum_{P}^{M} -log\left(\frac{e^{s_p}}{\sum_j^c e^{s_j}}\right)$$

Where each ***sp*** in M is the CNN score for each positive class. A scaling factor ***1/M*** to make the loss invariant to the number of positive classes, which may be different per sample. The gradient has different expressions for positive and negative classes. For positive classes:

$$\frac{}{\partial_{s_{p_i}}}\left(\frac{1}{M}\sum_{P}^{M} -log\left(\frac{e^{s_p}}{\sum_j^c e^{s_j}}\right)\right) = \frac{1}{M}\left(\left(\frac{e^{s_{pi}}}{\sum_j^c e^{s_j}} - 1\right) + (M-1)\left(\frac{e^{s_{pi}}}{\sum_j^c e^{s_j}}\right)\right)$$

Where ***s pi*** is the score of any positive class.
For negative classes:

$$\frac{\partial}{\partial_{s_n}}\left(\frac{1}{M}\sum_{P}^{M} -log\left(\frac{e^{s_p}}{\sum_j^c e^{s_j}}\right)\right) = \frac{e^{s_n}}{\sum_j^c e^{s_j}}$$

✓ **Metrics:** The primary evaluation metric used during training and validation is accuracy. Accuracy represents the proportion of correctly classified samples out of the total number of samples. It is a standard metric for classification tasks and provides a measure of how well the model is performing across all classes.

These choices are standard for multi-class image classification tasks. The Adam optimizer is employed for gradient descent, and categorical cross-entropy is used as the loss function to guide the model towards the correct class probabilities. The training progress is monitored using accuracy as the primary metric, reflecting the model's ability to correctly classify images during both training and validation phases

3.10. Evaluation Methods

In the provided code, various evaluation methods are employed to assess the performance of the Convolutional Neural Network (CNN) model for image classification. The training process utilizes an `ImageDataGenerator` to augment the training data, enhancing the model's ability to generalize to unseen examples. The `fit` method is then utilized to train the model on the augmented data, and the training history is visualized through plots that depict changes in accuracy and loss over epochs, providing insights into the model's convergence and potential overfitting or underfitting.

After training, the model's performance is evaluated on the test set using the `evaluate` method, which computes and prints the test accuracy and loss. This metric offers a quantitative measure of the model's accuracy on unseen data. Additionally, the training history is further visualized through plots displaying the model's accuracy and loss on both the training and validation sets. These plots aid in assessing the model's generalization to new data and detecting potential issues such as overfitting.

The model also includes a visual demonstration of the model's predictions by randomly selecting an image from the validation set, making predictions, and displaying both the original and predicted classes. This qualitative evaluation method provides an intuitive understanding of the model's ability to correctly classify images.

In summary, the combination of quantitative metrics, visualizations, and qualitative demonstrations offers a comprehensive evaluation of the CNN model's training dynamics, generalization performance, and predictive accuracy on both individual images and the overall test set.

CHAPTER FOUR

EXPERIMENT AND DISCUSSION

4.1. Experimental Design

4.2. Overview of the Experimental Setup

1. Study Area

a. Geographic Location: The study encompasses a diverse range of geographic locations, with latitude and longitude collected randomly from various sources available on the internet. Locations span different continents and regions to ensure a broad representation of soybean cultivation environments.

b. Climate Conditions

Temperature & Precipitation: Temperature data were sourced randomly, including measurements taken during both day and night. Daytime temperatures reflect conditions during active plant growth, while nighttime temperatures capture variations in cooler periods.

The temperature range covers various climatic zones, from temperate to tropical, providing a comprehensive understanding of how weed identification in soybean crops responds to different thermal regimes.

Precipitation data were collected to incorporate regions with varying rainfall patterns. This includes both dry and wet climates, allowing for an examination of how differing moisture levels impact weed prevalence in soybean fields.

2. Soil Characteristics

Texture & pH: Soil texture information is obtained from diverse locations, encompassing sandy, loamy, and clayey soils. This variability in texture is crucial for assessing how weed species interact with different soil types in soybean cultivation.

The pH levels of the soil are randomly sampled, covering a range from acidic to alkaline. This diversity in soil pH provides insights into how weed communities may be influenced by soil acidity or alkalinity in soybean fields.

Certainly, let's expand on the specifics of soybean varieties, weed species targeted, and growth stages observed in your experiment:

B. Key Variables and Factors Considered

1. Soybean Varieties

a. Selection Criteria: Various soybean varieties were intentionally chosen to represent a diverse set of genetic traits and characteristics. This includes factors such as resistance to pests, adaptation to different climates, and varying growth habits.

b. Early Growth Stage Emphasis: The primary focus of the study is on the early stages of soybean growth, typically from emergence to the vegetative stages. This stage is crucial for understanding the interaction between soybean plants and emerging weed competition.

c. Varietal Characteristics: Each soybean variety selected has distinct attributes, including growth rate, leaf structure, and canopy development. These variations contribute to the overall understanding of how different soybean traits may influence weed competition.

2. Weed Species Targeted

a. Broadleaf Weed Selection: The study narrowed its focus to broadleaf weed types, recognizing their significant impact on soybean crops during the early growth stages. This targeted approach helps in understanding the specific challenges posed by broadleaf weeds in soybean cultivation.

b. Diversity within Broadleaf Weeds: While the study focuses on broadleaf weeds, efforts were made to include a variety of these weeds to capture the diversity within this group. This encompasses different species with varying growth habits, leaf structures, and competitive strategies against soybean.

3. Growth Stages Observed

a. **Parallel Growth Trajectories**: The growth stages of soybean were intentionally aligned with the growth stages of selected broadleaf weed species. This approach facilitates a comparative analysis, allowing for the identification of critical growth stages where weed competition may be most pronounced.

b. Identification of Critical Points: Specific attention was given to growth stages where soybean and broadleaf weeds exhibit similarities or overlap. This includes stages such as germination, emergence, and early vegetative growth, which are pivotal in understanding the dynamics of weed competition.

By concentrating on early growth stages of soybean, selecting broadleaf weed types, and aligning growth stages for comparison, your study gains a focused perspective on the critical junctures where weed competition is likely to have a substantial impact on soybean crops. This targeted approach allows for a nuanced understanding of the interactions between soybean varieties and broadleaf weed species during their early developmental phases.

4.3. Model selection

In this thesis, we used a Convolutional Neural Network (CNN) as the cornerstone of my approach to weed and crop classification. The decision to employ a CNN stem from its proven efficacy in handling image classification tasks, particularly its capacity to autonomously learn intricate hierarchical features from input images. The utilization of the Keras library provided a robust and flexible platform for implementing this specific CNN architecture, allowing for seamless integration and efficient model development.

The CNN architecture selected for this study was meticulously crafted to cater specifically to the nuances of crop images. Understanding the inherent complexity and variability in agricultural imagery, the model architecture was designed to adapt and capture diverse patterns and relationships within the visual data. The chosen architecture incorporates a series of convolutional and max-pooling layers to systematically extract essential features, followed by fully connected layers that enable the model to make informed predictions based on the learned representations.

Activation functions, particularly the rectified linear unit (ReLU), were strategically applied throughout the model to introduce non-linearity and enhance the network's ability to capture intricate relationships within the image data. The choice of activation functions played a pivotal role in fostering the model's capacity to discern and represent complex features associated with different crop types.

In the context of model compilation, the Adam optimizer was chosen for its adaptability and efficiency in handling large-scale datasets. The categorical crossentropy loss function was employed to guide the model in minimizing the disparity between predicted probabilities and true class labels, aligning with the multiclass nature of the crop classification task.

4.4. Comparison of Pre-trained CNN Model

In this comprehensive experiment, we explored and compared seven distinct approaches to image classification using convolutional neural networks (CNNs). My objective was to investigate the impact of various strategies, including leveraging pre-trained models, customizing network architectures, fine-tuning parameters, and augmenting training data. Each approach was carefully designed to understand its influence on model performance and generalization across a diverse set of image classes.

Transfer Learning with Pre-trained Models:

Utilized popular pre-trained models such as VGG19 and ResNet50, to leverage learned features. Investigated the effectiveness of transfer learning in terms of accuracy and convergence speed.

4.4.1 VGG 19

VGG19 proposed by Simonyan and Zisserman (2014) is a convolutional neural network that comprises 19 layers with 16 convolution layers and 3 fully connected to classify the images into 1000 object categories. VGG19 is trained on the ImageNet database that contains a million images of 1000 categories. It is a very popular method for image classification due to the use of multiple 3 × 3 filters in each convolutional layer. The architecture of VGG19 is shown in Fig. This shows that 16 convolutional layers are used for feature extraction and the next 3 layers work for classification [51].

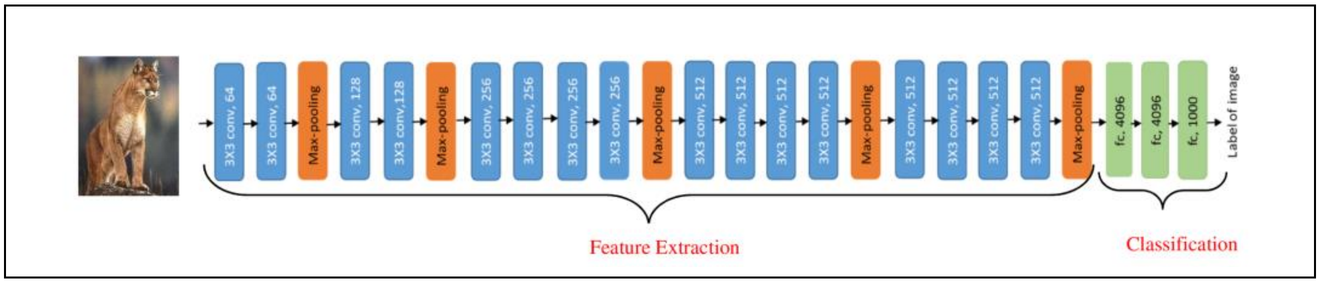
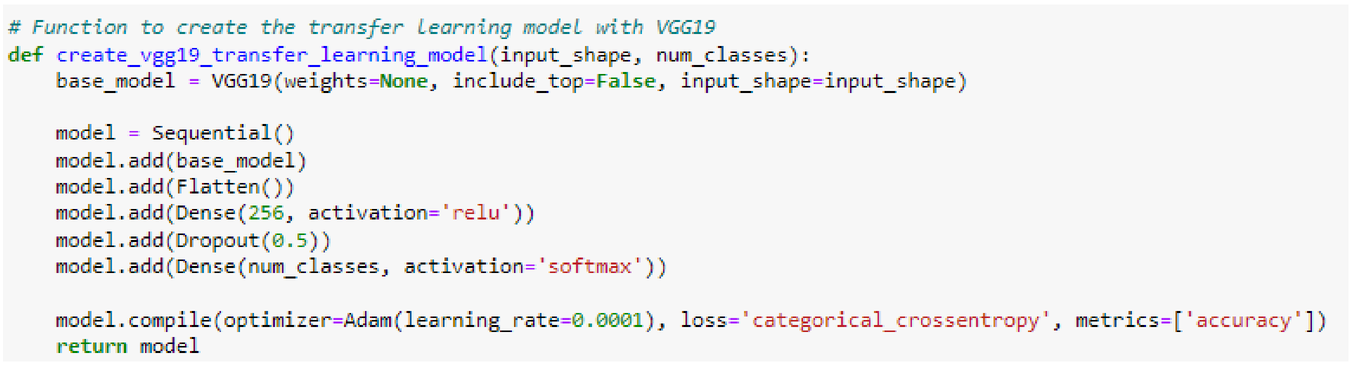


Fig 4- 1 Architecture of VGG19 model [57]

In this study several attempts have been made one implements a transfer learning approach using the VGG19 model for a specific image classification task. Here's a result:

Creating Transfer Learning Model using VGG19:

A function (create_vgg19_transfer_learning_model) is defined to create a transfer learning model based on VGG19. This model consists of the VGG19 base (excluding the top layers), followed by additional layers for the specific task. The model is compiled with an Adam optimizer, categorical crossentropy loss, and accuracy as the metric.



```python
# Function to create the transfer learning model with VGG19
def create_vgg19_transfer_learning_model(input_shape, num_classes):
    base_model = VGG19(weights=None, include_top=False, input_shape=input_shape)

    model = Sequential()
    model.add(base_model)
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Fig 4- 2 Creating Transfer Learning Model using VGG19

A training function (train_vgg19_transfer_model) is provided, utilizing the created VGG19 transfer learning model. It uses an ImageDataGenerator for data augmentation and loads images from specified directories for training, validation, and testing. The model is trained for a specified number of epochs, and its performance is evaluated on the test set.

VGG19 Weights	[Link] VGG19
Number of Classes	4
Model Input Shape	(64, 64, 3)
Batch Size	20
Number of Epochs	10
Test Loss	0.2363
Test Accuracy	90.78%

Table 4- 1 Parameters for transfer learning using VGG19

4.4.2. ResNet50 (Residual Network)

ResNet50, short for Residual Network with 50 layers, is a deep convolutional neural network architecture that belongs to the ResNet family. ResNet was introduced by Microsoft to address the vanishing gradient problem that often occurs in very deep neural networks. The key innovation in ResNet is the use of residual learning, where shortcut connections (skip connections) allow the gradient to bypass certain layers during training [51]. This model was immensely successful, as can be ascertained from the fact that its ensemble won the top position at the ILSVRC 2015 classification competition with an error of only 3.57% [58].

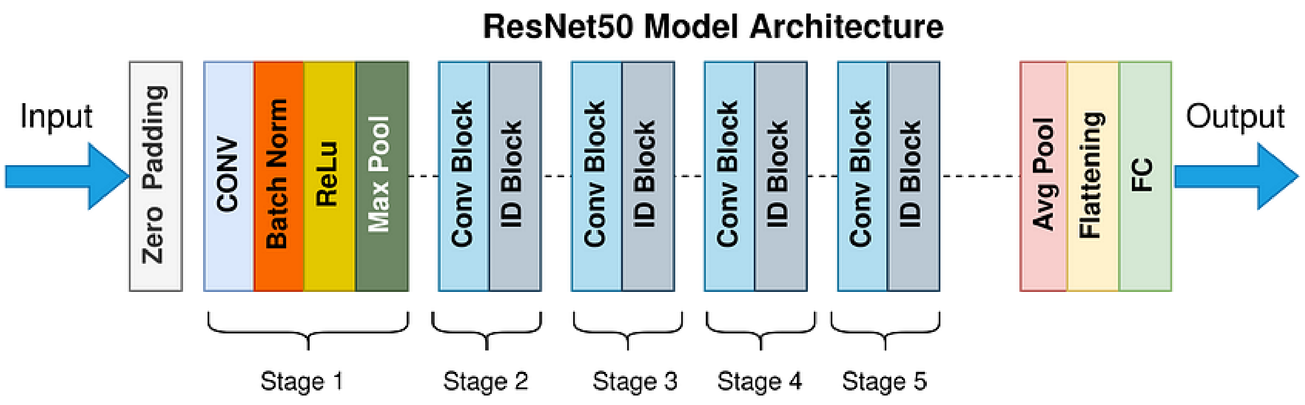


Fig 4- 3 Resnet50 Model architecture [58]

The other experiment transfer learning approach using the ResNet50 model for a specific image classification task. Here's a result:

The code implements transfer learning using the ResNet50 model for a classification task on a dataset with four classes. The ResNet50 model is pre-trained on ImageNet and fine-tuned for the specific classification task.

A function (create_resnet_transfer_learning_model) creates a transfer learning model using ResNet50 as the base model. Adds additional layers such as Flatten, Dense, Dropout for custom classification.

A function (train_resnet50_transfer_model) is implemented to train the ResNet50 transfer learning model. Training involves using data generators for training, validation, and testing. Model performance metrics such as loss, accuracy, and a classification report are printed.

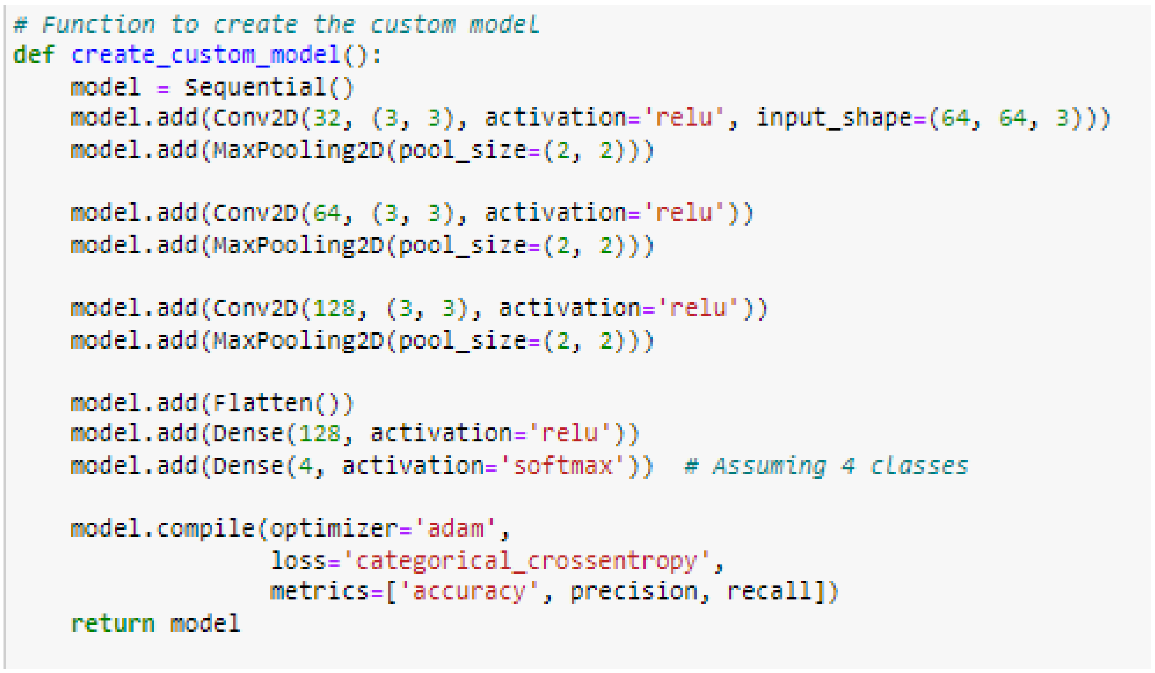
Parameter	Value
Learning Rate	0.0001
Dropout Rate	0.5
Number of Classes	4
Input Shape	(64, 64, 3)
Batch Size	20
Test Loss	0.3428 %
Test Accuracy	0.8755 %

Table 4- 2 Parameters and result transfer learning using VGG19

4.4.3. Convolutional neural network (CNN) using the TensorFlow and Keras libraries for image classification with different parameters and less layers

The other experiment is evaluating a convolutional neural network (CNN) using the TensorFlow and Keras libraries for image classification.

The architecture of the convolutional neural network (CNN) is designed to capture intricate features in the agricultural land cover images. The custom model consists of three convolutional layers, each followed by max-pooling, and concludes with densely connected layers. Custom metric functions, including precision and recall, are integrated to provide a comprehensive evaluation of the model's performance.



```
# Function to create the custom model
def create_custom_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(4, activation='softmax'))  # Assuming 4 classes

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy', precision, recall])
    return model
```

Data augmentation is a crucial aspect of training a robust model. The" ImageDataGenerator" class from TensorFlow is leveraged to apply various transformations, including rotation, shifting, shearing, zooming, and horizontal flipping. Key parameters for augmentation include:

Rotation range: 20 degrees

Width shift range: 0.2

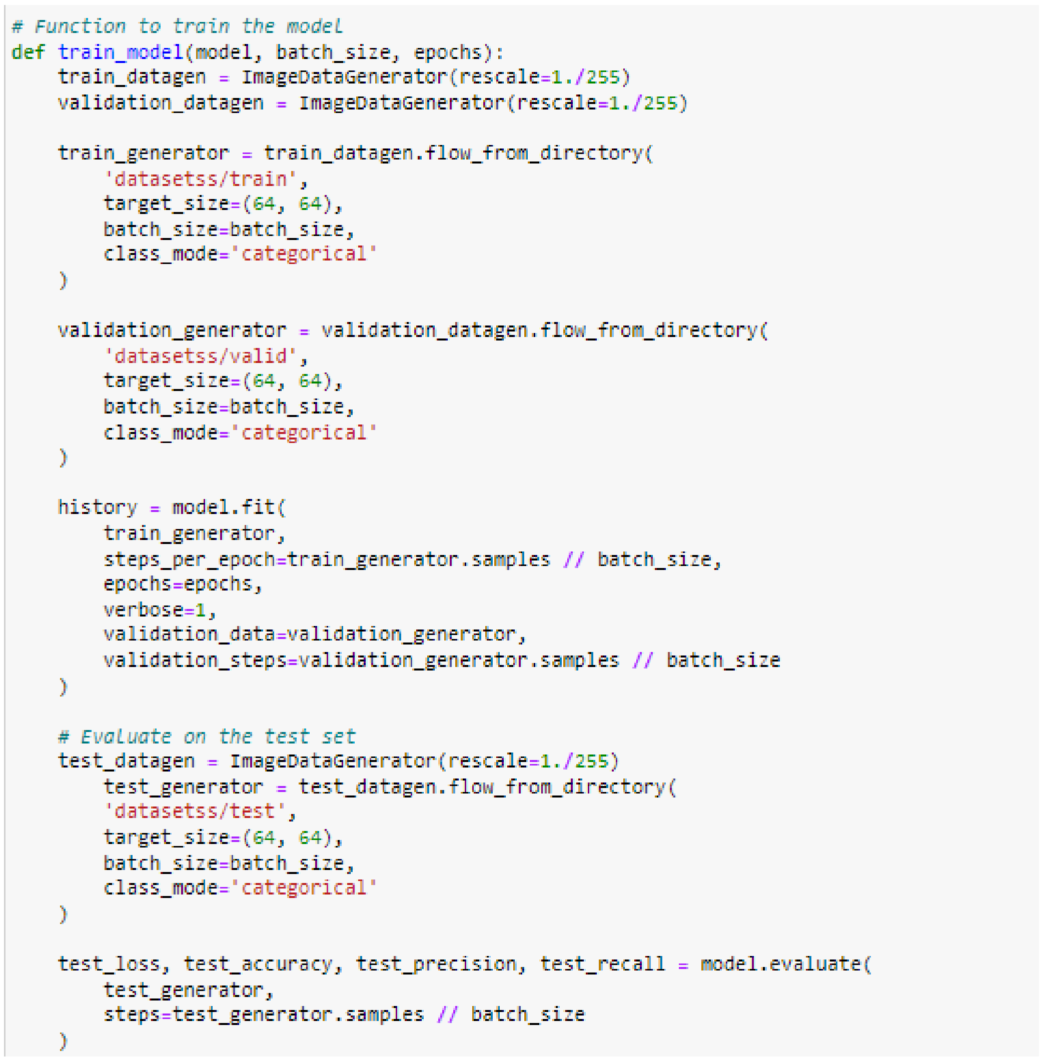
Height shift range: 0.2

Shear range: 0.2

Zoom range: 0.2

Horizontal flip: True

The model is trained using the training dataset, and the training history is recorded for subsequent analysis. Additionally, the validation set is employed to monitor the model's performance during training and prevent overfitting. After training completion, the model is evaluated on the previously unseen test set, and metrics such as accuracy, precision, and recall are computed.



```
# Function to train the model
def train_model(model, batch_size, epochs):
    train_datagen = ImageDataGenerator(rescale=1./255)
    validation_datagen = ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
        'datasetss/train',
        target_size=(64, 64),
        batch_size=batch_size,
        class_mode='categorical'
    )

    validation_generator = validation_datagen.flow_from_directory(
        'datasetss/valid',
        target_size=(64, 64),
        batch_size=batch_size,
        class_mode='categorical'
    )

    history = model.fit(
        train_generator,
        steps_per_epoch=train_generator.samples // batch_size,
        epochs=epochs,
        verbose=1,
        validation_data=validation_generator,
        validation_steps=validation_generator.samples // batch_size
    )

    # Evaluate on the test set
    test_datagen = ImageDataGenerator(rescale=1./255)
        test_generator = test_datagen.flow_from_directory(
        'datasetss/test',
        target_size=(64, 64),
        batch_size=batch_size,
        class_mode='categorical'
    )

    test_loss, test_accuracy, test_precision, test_recall = model.evaluate(
        test_generator,
        steps=test_generator.samples // batch_size
    )
```

Fig 4- 4 Monitoring validation during training

Parameter/Metric	**Value**
Training Set Size	9200 images (4 classes: broadleaf, grass, soybean, soil)
Validation Set Size	3067 images (4 classes)
Test Set Size	3069 images (4 classes)
Epochs	10
Batch Size	20
Metrics (Per Epoch)	Loss, Accuracy, Precision, Recall (Training)
Test Set Metrics	Test Loss: 0.1599, Test Accuracy: 0.9507, Test Precision: 0.9516, Test Recall: 0.9503
Metrics (Per Epoch)	Loss, Accuracy, Precision, Recall (Validation)

Table 4- 3 Augmentation techniques that have been used on proposed model

The time taken for each epoch and the entire training process may vary depending on the hardware used.

```
Found 9200 images belonging to 4 classes.
Found 3067 images belonging to 4 classes.
Epoch 1/10
460/460 [==============================] - 424s 920ms/step - loss: 0.4717 - accuracy: 0.8161 - precision: 0.8438 - recall: 0.77
07 - val_loss: 0.2819 - val_accuracy: 0.9026 - val_precision: 0.9214 - val_recall: 0.8824
Epoch 2/10
460/460 [==============================] - 284s 617ms/step - loss: 0.2753 - accuracy: 0.8960 - precision: 0.9086 - recall: 0.88
35 - val_loss: 0.2270 - val_accuracy: 0.9216 - val_precision: 0.9287 - val_recall: 0.9127
Epoch 3/10
460/460 [==============================] - 317s 690ms/step - loss: 0.2245 - accuracy: 0.9191 - precision: 0.9272 - recall: 0.91
00 - val_loss: 0.1764 - val_accuracy: 0.9330 - val_precision: 0.9413 - val_recall: 0.9239
Epoch 4/10
460/460 [==============================] - 387s 841ms/step - loss: 0.1803 - accuracy: 0.9347 - precision: 0.9416 - recall: 0.92
70 - val_loss: 0.1668 - val_accuracy: 0.9435 - val_precision: 0.9541 - val_recall: 0.9297
Epoch 5/10
460/460 [==============================] - 125s 272ms/step - loss: 0.1495 - accuracy: 0.9452 - precision: 0.9492 - recall: 0.94
00 - val_loss: 0.1412 - val_accuracy: 0.9471 - val_precision: 0.9521 - val_recall: 0.9415
Epoch 6/10
460/460 [==============================] - 34s 73ms/step - loss: 0.1301 - accuracy: 0.9513 - precision: 0.9549 - recall: 0.9468
- val_loss: 0.1477 - val_accuracy: 0.9412 - val_precision: 0.9455 - val_recall: 0.9379
Epoch 7/10
460/460 [==============================] - 33s 71ms/step - loss: 0.0989 - accuracy: 0.9635 - precision: 0.9664 - recall: 0.9605
- val_loss: 0.1495 - val_accuracy: 0.9448 - val_precision: 0.9483 - val_recall: 0.9408
Epoch 8/10
460/460 [==============================] - 33s 73ms/step - loss: 0.0781 - accuracy: 0.9712 - precision: 0.9729 - recall: 0.9697
- val_loss: 0.1479 - val_accuracy: 0.9464 - val_precision: 0.9504 - val_recall: 0.9451
Epoch 9/10
460/460 [==============================] - 33s 72ms/step - loss: 0.0646 - accuracy: 0.9770 - precision: 0.9786 - recall: 0.9751
- val_loss: 0.1911 - val_accuracy: 0.9310 - val_precision: 0.9346 - val_recall: 0.9294
Epoch 10/10
460/460 [==============================] - 33s 71ms/step - loss: 0.0385 - accuracy: 0.9865 - precision: 0.9871 - recall: 0.9858
- val_loss: 0.1347 - val_accuracy: 0.9569 - val_precision: 0.9586 - val_recall: 0.9559
Found 3069 images belonging to 4 classes.
153/153 [==============================] - 127s 833ms/step - loss: 0.1599 - accuracy: 0.9507 - precision: 0.9516 - recall: 0.95
03
Test Loss: 0.1599
Test Accuracy: 0.9507
Test Precision: 0.9516
Test Recall: 0.9503
Training completed successfully.
```

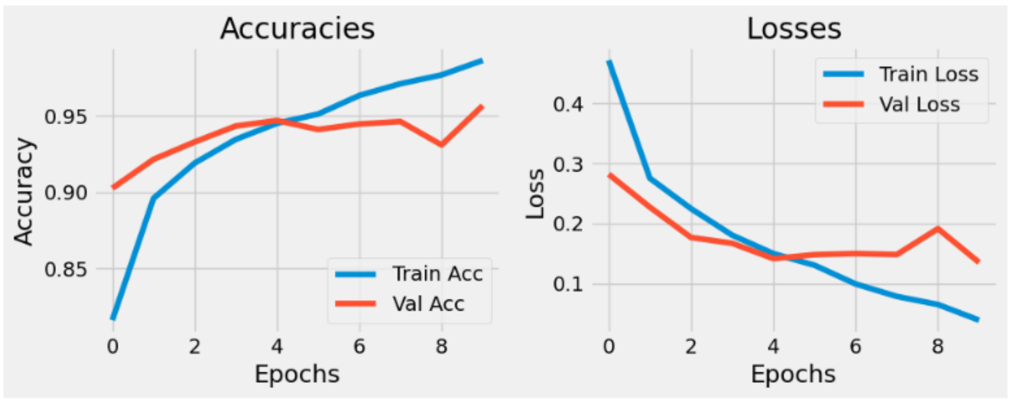


Fig 4- 5 Accuracy and loss of training and validation proposed model

The experimental code focuses on employing data augmentation techniques for crop classification using a convolutional neural network (CNN). The dataset consists of four classes—'broadleaf,' 'grass,' 'soil,' and 'soybean'—with images organized in separate directories for each class. The `DataAugmentation` class encapsulates image processing methods, including rotation and flipping, and these augmentations are applied to each image in the dataset, with the augmented images stored in a new directory ('augmented_images').

The training process involves utilizing the augmented training data and validating on the original validation set. Training history, encompassing accuracy and loss, is saved, and the script allows loading a pre-trained model if available. The trained model is evaluated on the test set, and accuracy is displayed. The script further visualizes training history using matplotlib and showcases a random image from the validation set to demonstrate the model's prediction.

The experimental setup underscores the significance of data augmentation in enhancing model generalization for crop classification tasks. The comprehensive pipeline covers preprocessing, model training, and evaluation, providing a robust foundation for further experimentation and analysis in agricultural image classification scenarios.

Parameter/Metric	**Value**
Training Set Size	9200 images (4 classes: broadleaf, grass, soybean, soil)
Validation Set Size	3067 images (4 classes)
Test Set Size	3069 images (4 classes)
Number of Epochs	20
Batch Size	20
Image Dimensions	200x200x3
Model Architecture	CNN with Conv2D, MaxPooling2D, Flatten, Dense layers
Dropout Rate	0.5
Optimizer	Adam
Loss Function	Categorical Crossentropy
Learning Rate	Default (Adam optimizer)
Final Training Loss & Accuracy	0.1297 - 94.00%
Total params	13,628,484

Table 4- 4 Experiment input and result

In the subsequent phases of experimentation, the focus shifted towards refining the performance of the neural network through systematic adjustments to data augmentation parameters and variations in batch size. The augmentation parameters, which determine the extent of image transformations applied during training, were systematically tweaked to explore their impact on the model's ability to generalize from the training data to unseen samples.

The augmentation parameters encompassed a range of transformations, including rotation, width and height shifts, shearing, zooming, and horizontal flipping. By modifying these parameters, the aim was to enhance the robustness of the model by exposing it to a diverse set of artificially generated images. The goal was to evaluate how well the model could handle variations in input data and whether these augmentations contributed to improved overall performance.

Simultaneously, the batch size, representing the number of training samples utilized in one iteration, was varied to examine its influence on the training dynamics. Different batch sizes were employed to assess trade-offs between computational efficiency and the model's ability to learn from the data effectively. This exploration sought to identify optimal configurations that struck a balance between computational resource utilization and training effectiveness.

These adjustments and variations in augmentation parameters and batch size were part of a comprehensive strategy to iteratively refine the model's architecture. The overarching objective was to enhance its capability to accurately classify images across diverse scenarios and improve its generalization performance on unseen data. Each experimental iteration aimed to uncover insights into the nuanced interplay between these hyperparameters and the neural network's learning dynamics. Here is all the experiments result:

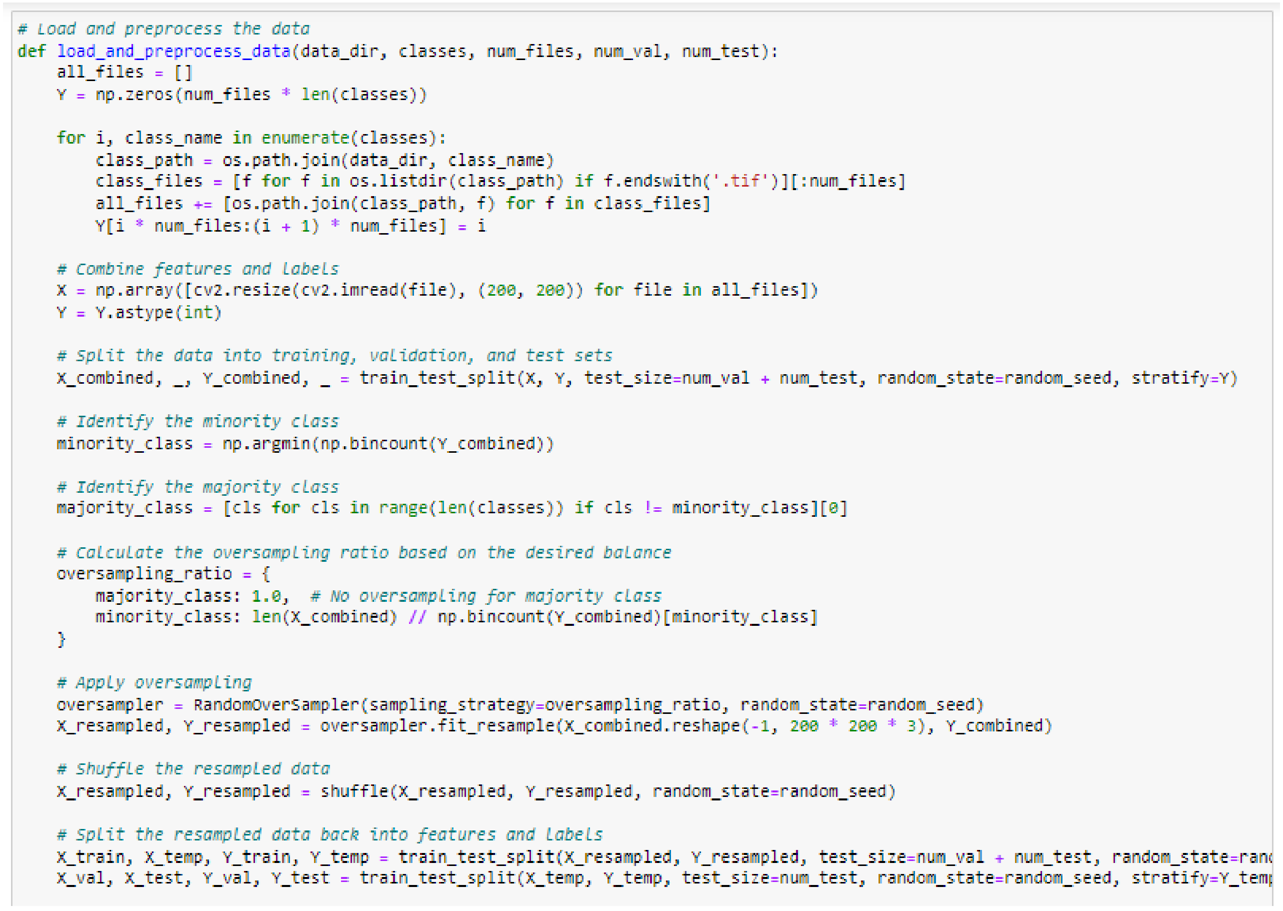
	Transfer Learning Model	Architecture	Image Size	Data Augmentation	Normalization	Optimizer	Loss Function	Training Epochs	Test Accuracy
1	None	Convolutional Neural Network	(200, 200, 3)	Yes (ImageDataGenerator)	Yes (MinMax Scaling)	Adam	Categorical Crossentropy	20	94.00 %
2	None	Convolutional Neural Network	(200, 200, 3)	Yes (ImageDataGenerator)	Yes (Minmax Scaling)	Adam	Categorical Crossentropy	15	94.67 %
3	None	Convolutional Neural Network	(200, 200, 3)	Yes (ImageDataGenerator)	Yes (MinMax Scaling)	Adam	Categorical Crossentropy	10	92.00 %
4	None	Convolutional Neural Network	(64, 64, 3)	Yes (ImageDataGenerator)	Yes (MinMax Scaling)	Adam	Categorical Crossentropy	10	95.07 %
5	CNN with Data Augmentation	Convolutional Neural Network	(200, 200, 3)	Yes (ImageDataGenerator)	Yes (MinMax Scaling)	Adam	Categorical Crossentropy	20	95.00 %

Table 4- 5 Convolutional Neural Network Experimental results

4.5. Strategies for Addressing Class Imbalance

The datasets exhibit class imbalance, with significant variations in the number of samples across the four classes: broadleaf (1191 samples), grass (3520 samples), soil (3249 samples), and soybean (7376 samples). This imbalance poses a challenge during training, as the model may become biased towards the majority class (soybean). While the weed class, represented by broadleaf, is crucial for accurate classification, its lower representation in the dataset can hinder the model's ability to effectively learn its features.

In an attempt to address this issue, the training section employs a strategy to balance the dataset by setting the number of files for each class, `num_file`, to 1100, multiplied by a scaling factor `m`. The scaling factor, set to 1 (`m = 1`), results in a balanced subset of the data, with 1100 images from each class. However, as highlighted, this may not be sufficient for training a robust model, especially given the significant disparity in class sizes.

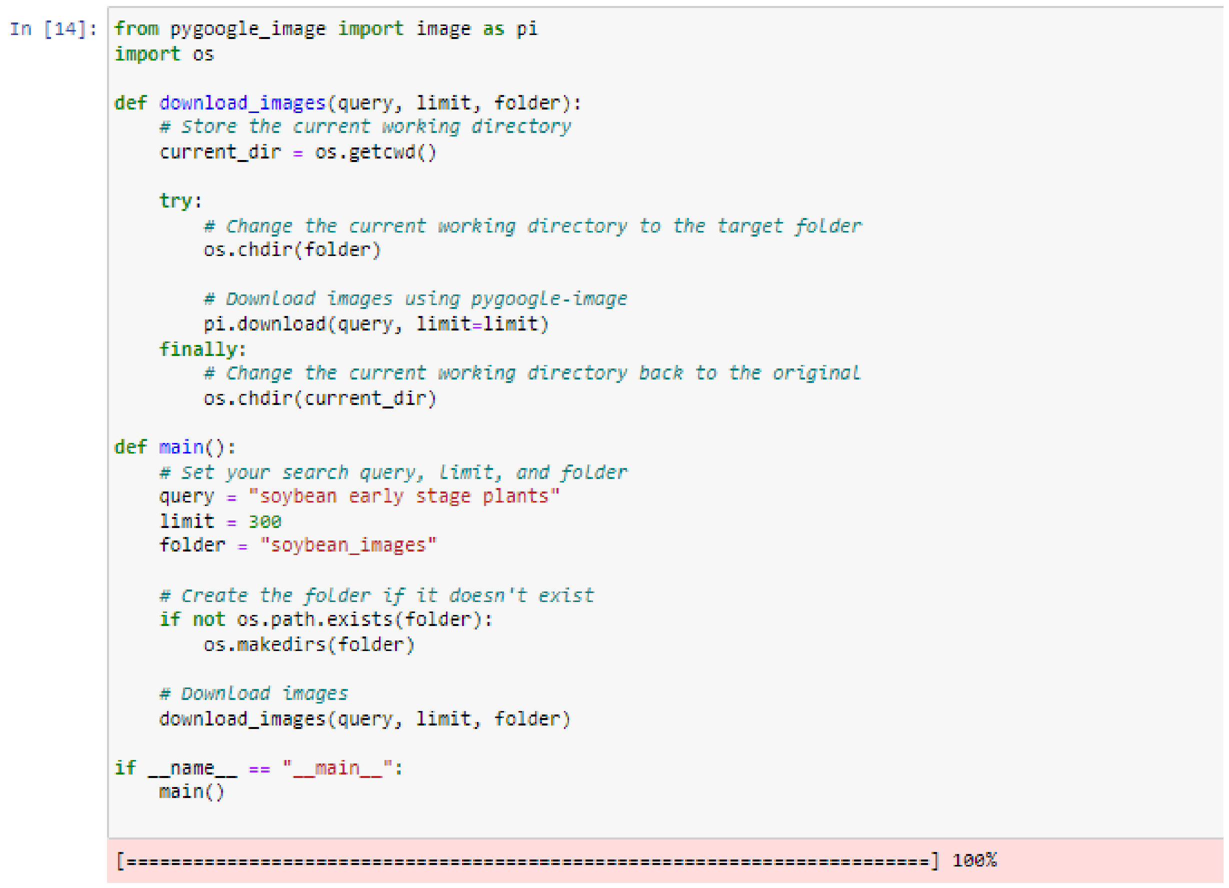


```
# Load and preprocess the data
def load_and_preprocess_data(data_dir, classes, num_files, num_val, num_test):
    all_files = []
    Y = np.zeros(num_files * len(classes))

    for i, class_name in enumerate(classes):
        class_path = os.path.join(data_dir, class_name)
        class_files = [f for f in os.listdir(class_path) if f.endswith('.tif')][:num_files]
        all_files += [os.path.join(class_path, f) for f in class_files]
        Y[i * num_files:(i + 1) * num_files] = i

    # Combine features and labels
    X = np.array([cv2.resize(cv2.imread(file), (200, 200)) for file in all_files])
    Y = Y.astype(int)

    # Split the data into training, validation, and test sets
    X_combined, _, Y_combined, _ = train_test_split(X, Y, test_size=num_val + num_test, random_state=random_seed, stratify=Y)

    # Identify the minority class
    minority_class = np.argmin(np.bincount(Y_combined))

    # Identify the majority class
    majority_class = [cls for cls in range(len(classes)) if cls != minority_class][0]

    # Calculate the oversampling ratio based on the desired balance
    oversampling_ratio = {
        majority_class: 1.0,  # No oversampling for majority class
        minority_class: len(X_combined) // np.bincount(Y_combined)[minority_class]
    }

    # Apply oversampling
    oversampler = RandomOverSampler(sampling_strategy=oversampling_ratio, random_state=random_seed)
    X_resampled, Y_resampled = oversampler.fit_resample(X_combined.reshape(-1, 200 * 200 * 3), Y_combined)

    # Shuffle the resampled data
    X_resampled, Y_resampled = shuffle(X_resampled, Y_resampled, random_state=random_seed)

    # Split the resampled data back into features and labels
    X_train, X_temp, Y_train, Y_temp = train_test_split(X_resampled, Y_resampled, test_size=num_val + num_test, random_state=ranc
    X_val, X_test, Y_val, Y_test = train_test_split(X_temp, Y_temp, test_size=num_test, random_state=random_seed, stratify=Y_temp
```

Fig 4- 6 Oversampling to balanced datasets

To mitigate the impact of imbalanced classes, the code incorporates oversampling for some of the underrepresented classes. This strategy involves generating synthetic samples for minority classes to ensure a more equitable representation during training. Although oversampling helps address the imbalance issue, it introduces new challenges and considerations in terms of model generalization and potential overfitting.

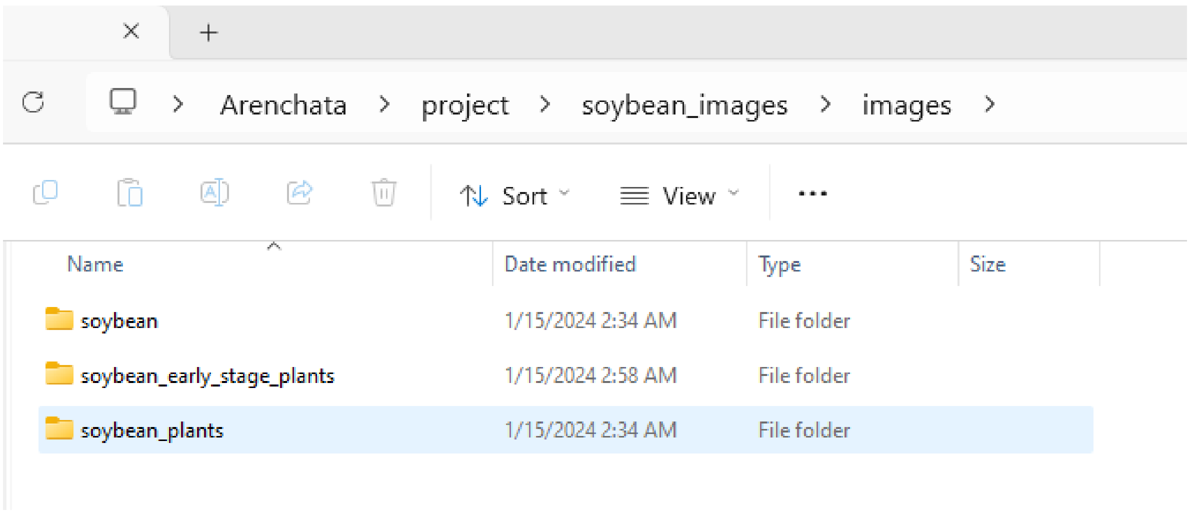
Moreover, to augment the dataset further and enhance model robustness, the code describes a custom script that leverages Google Images for additional image acquisition. This script allows the automatic retrieval and downloading of images based on user-defined queries. The acquired images undergo preprocessing using various techniques, such as edge detection, color segmentation, erosion, dilation, and Gaussian blur. These preprocessing steps aim to enhance the diversity and quality of the training data, contributing to a more comprehensive and effective training process.



```
In [14]: from pygoogle_image import image as pi
         import os

         def download_images(query, limit, folder):
             # Store the current working directory
             current_dir = os.getcwd()

             try:
                 # Change the current working directory to the target folder
                 os.chdir(folder)

                 # Download images using pygoogle-image
                 pi.download(query, limit=limit)
             finally:
                 # Change the current working directory back to the original
                 os.chdir(current_dir)

         def main():
             # Set your search query, limit, and folder
             query = "soybean early stage plants"
             limit = 300
             folder = "soybean_images"

             # Create the folder if it doesn't exist
             if not os.path.exists(folder):
                 os.makedirs(folder)

             # Download images
             download_images(query, limit, folder)

         if __name__ == "__main__":
             main()

         [==========================================================================] 100%
```

Fig 4- 7 Additional image acquisition

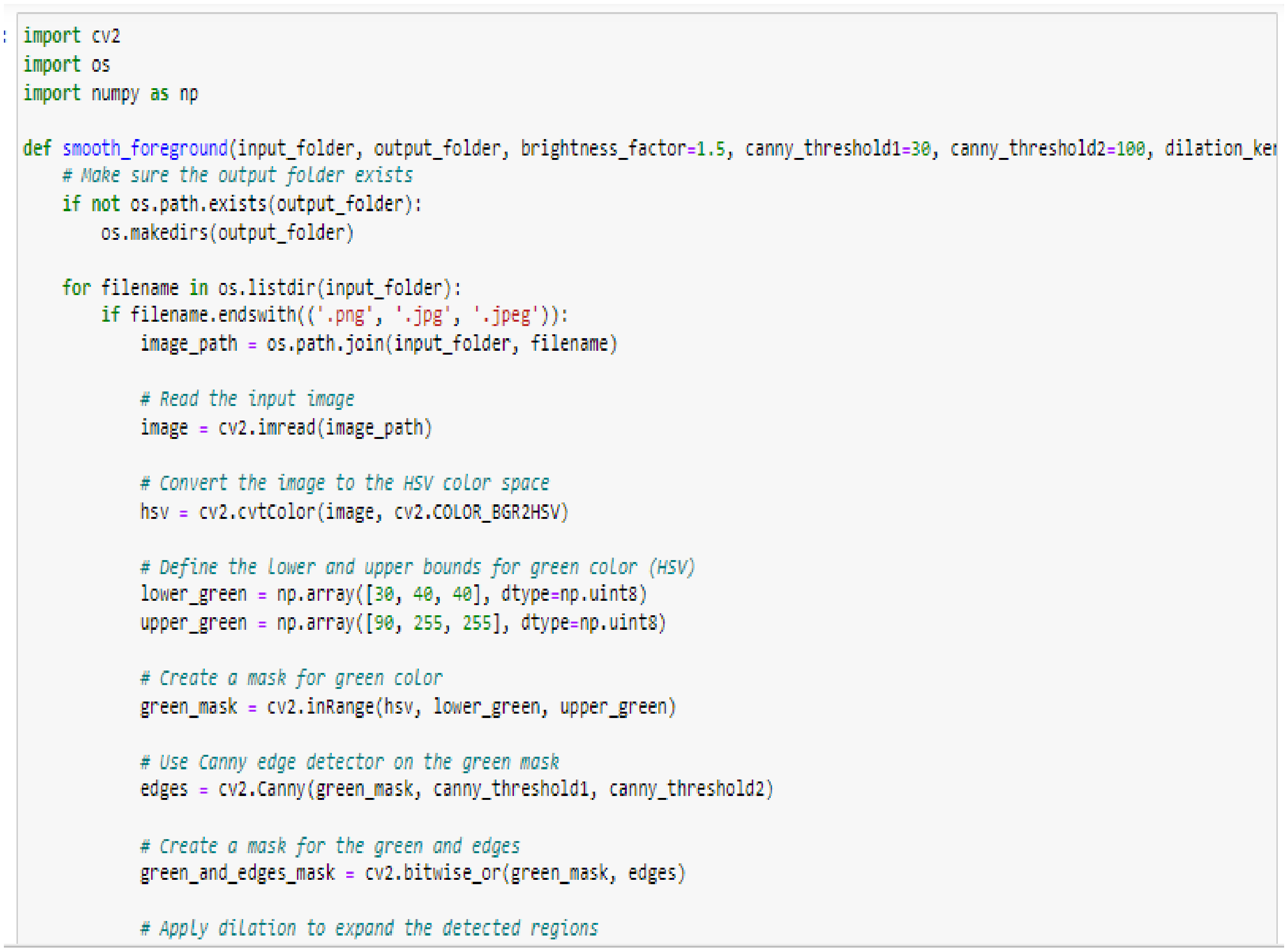
The provided Python script utilizes the `pygoogle_image` library to download a specified number of images related to a particular search query. In this case, the query is set to "soybean early-stage plants," with a limit of 300 images to be downloaded. The images are stored in a folder named "soybean_images." The script encapsulates this functionality within a `download_images` function, allowing for flexibility in changing the search query, download limit, and target folder. The `main` function sets the parameters and ensures the creation of the target folder if it does not already exist. Finally, the script is executed when the file is run, initiating the image download process based on the specified query, limit, and folder.



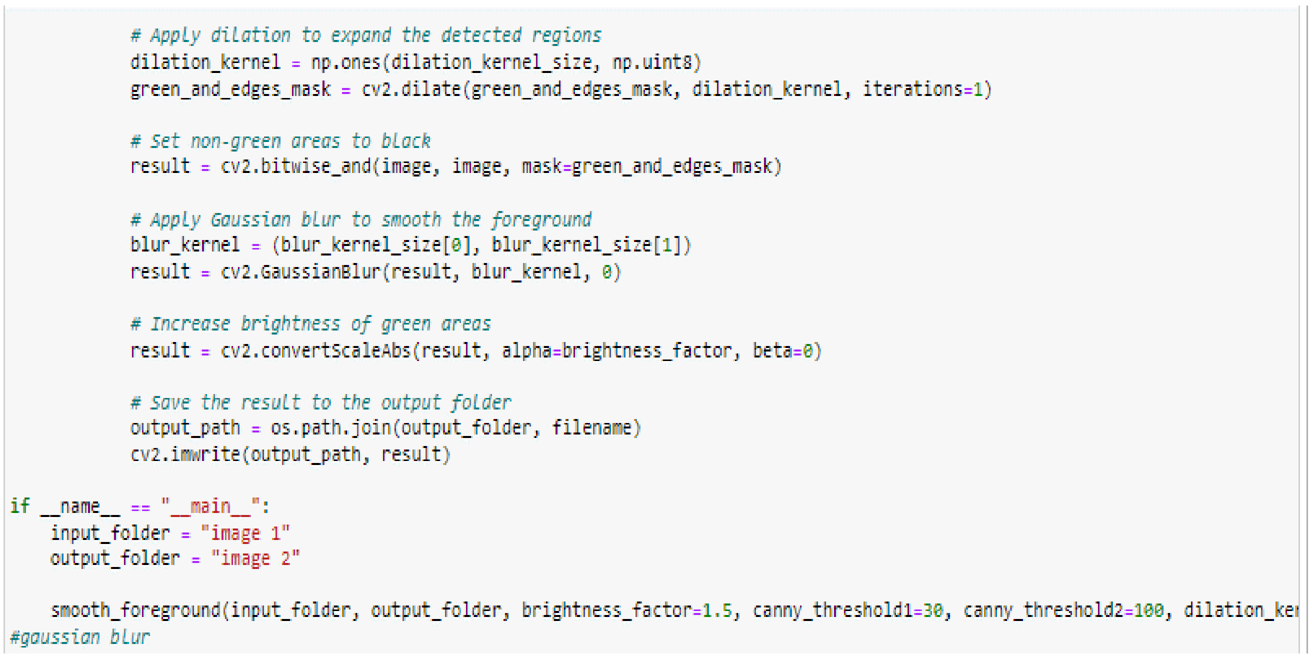
Name	Date modified	Type	Size
soybean	1/15/2024 2:34 AM	File folder	
soybean_early_stage_plants	1/15/2024 2:58 AM	File folder	
soybean_plants	1/15/2024 2:34 AM	File folder	

In summary, the paragraphs elucidate the challenges posed by imbalanced datasets, the attempts made to balance class representation during training, and the supplementary measures taken to augment the dataset through web scraping and preprocessing techniques. These strategies collectively contribute to addressing the challenges associated with imbalanced datasets and aim to improve the overall performance of the classification model, especially in the context of weed detection.

After the images are downloaded using the provided script, they undergo a series of image preprocessing steps to enhance their quality and prepare them for subsequent analysis. The preprocessing pipeline includes several key techniques:



```python
import cv2
import os
import numpy as np

def smooth_foreground(input_folder, output_folder, brightness_factor=1.5, canny_threshold1=30, canny_threshold2=100, dilation_ker
    # Make sure the output folder exists
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    for filename in os.listdir(input_folder):
        if filename.endswith(('.png', '.jpg', '.jpeg')):
            image_path = os.path.join(input_folder, filename)

            # Read the input image
            image = cv2.imread(image_path)

            # Convert the image to the HSV color space
            hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

            # Define the lower and upper bounds for green color (HSV)
            lower_green = np.array([30, 40, 40], dtype=np.uint8)
            upper_green = np.array([90, 255, 255], dtype=np.uint8)

            # Create a mask for green color
            green_mask = cv2.inRange(hsv, lower_green, upper_green)

            # Use Canny edge detector on the green mask
            edges = cv2.Canny(green_mask, canny_threshold1, canny_threshold2)

            # Create a mask for the green and edges
            green_and_edges_mask = cv2.bitwise_or(green_mask, edges)

            # Apply dilation to expand the detected regions
```



```python
            # Apply dilation to expand the detected regions
            dilation_kernel = np.ones(dilation_kernel_size, np.uint8)
            green_and_edges_mask = cv2.dilate(green_and_edges_mask, dilation_kernel, iterations=1)

            # Set non-green areas to black
            result = cv2.bitwise_and(image, image, mask=green_and_edges_mask)

            # Apply Gaussian blur to smooth the foreground
            blur_kernel = (blur_kernel_size[0], blur_kernel_size[1])
            result = cv2.GaussianBlur(result, blur_kernel, 0)

            # Increase brightness of green areas
            result = cv2.convertScaleAbs(result, alpha=brightness_factor, beta=0)

            # Save the result to the output folder
            output_path = os.path.join(output_folder, filename)
            cv2.imwrite(output_path, result)

if __name__ == "__main__":
    input_folder = "image 1"
    output_folder = "image 2"

    smooth_foreground(input_folder, output_folder, brightness_factor=1.5, canny_threshold1=30, canny_threshold2=100, dilation_ker
#gaussian blur
```

Fig 4- 8 Enhancing the downloaded images before training

The provided Python script implements a foreground smoothing technique using Gaussian blur on images with green elements. The purpose of this script is to enhance and smooth the green regions in images, creating visually appealing results. Here's an overview of the key functionalities:

The script takes images from an input folder ("image 1") and saves the processed images to an output folder ("image 2").

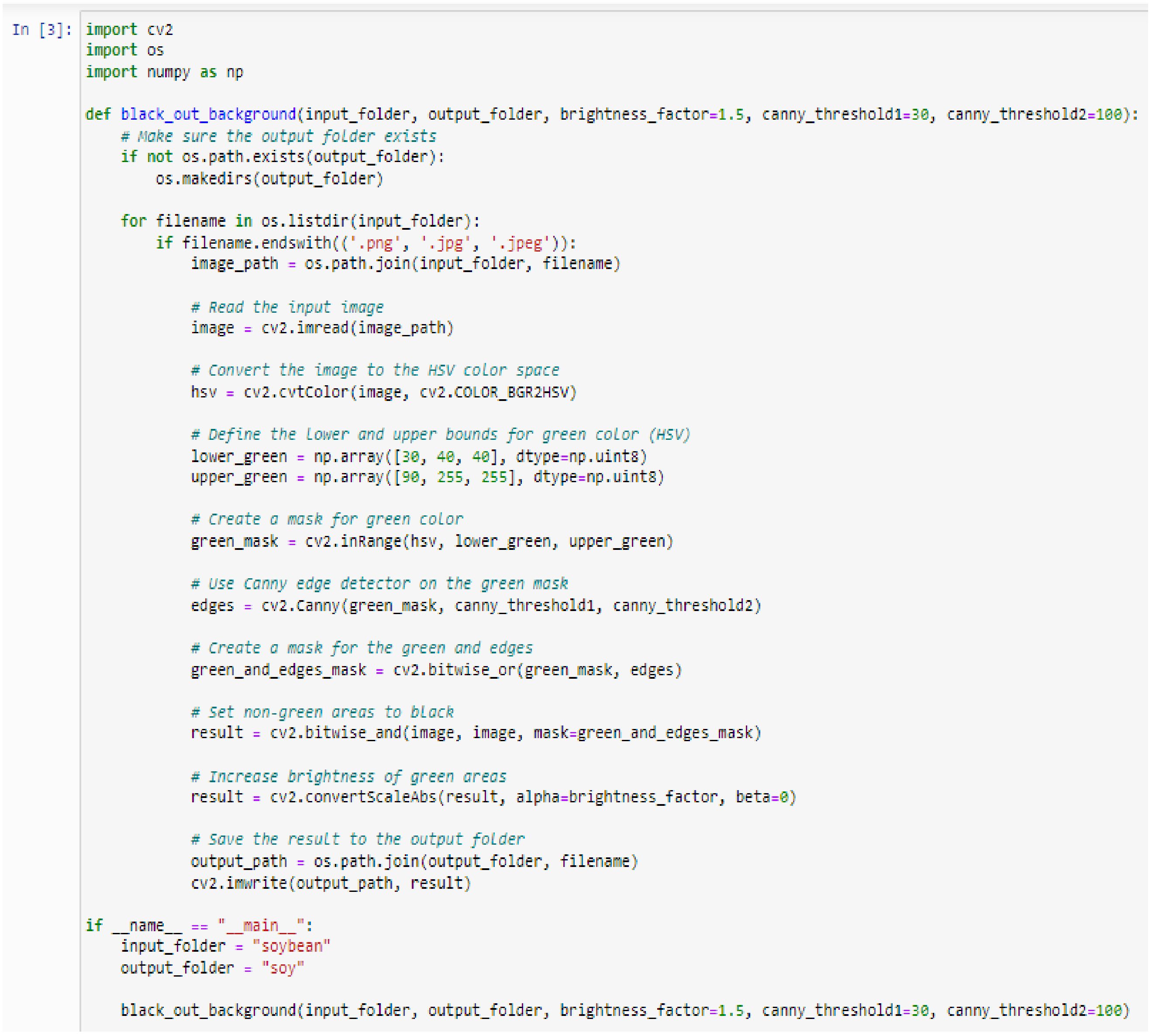
- Color Masking: The script converts each input image to the HSV color space and creates a mask to isolate green-colored regions based on predefined lower and upper bounds.
- Canny Edge Detection: The Canny edge detector is applied to the green mask, enhancing the edges of the green elements.
- Mask Combination: The script combines the green mask and edges mask using a bitwise OR operation, resulting in a mask that highlights both green areas and their edges.
- Dilation: A dilation operation is performed on the combined mask to expand and strengthen the detected regions.
- Foreground Extraction: The original image is then bitwise ANDed with the combined mask, effectively setting non-green areas to black and isolating the smoothed green foreground.
- Gaussian Blur: Finally, a Gaussian blur is applied to the extracted green foreground to achieve a smooth and visually pleasing result.
- Brightness Adjustment: The brightness of the green areas is increased by using the `cv2.convertScaleAbs` function, enhancing the overall appearance of the smoothed foreground.

The script can be customized by adjusting parameters such as brightness factor, Canny edge detection thresholds, dilation kernel size, and Gaussian blur kernel size. Overall, this script provides a versatile and automated approach to enhancing green elements in images while maintaining a smooth and aesthetically pleasing foreground.

The next provided Python script performs background removal and brightness adjustment on images containing green elements, with a focus on isolating soybean plants. The script utilizes computer vision techniques, including color masking, edge detection, and bitwise operations, to achieve the desired effect. Here's an overview of the key functionalities:

The script takes images from an input folder ("soybean") and saves the processed images to an output folder ("soy").

- Color Masking: Each input image is converted to the HSV color space, and a mask is created to isolate green-colored regions. The defined lower and upper bounds for green color in HSV ensure accurate detection.
- Canny Edge Detection: The Canny edge detector is applied to the green mask, enhancing the edges of the green elements. This step is crucial for capturing intricate details in the soybean plants.
- Mask Combination: The script combines the green mask and edges mask using a bitwise OR operation, creating a comprehensive mask that highlights both green areas and their edges.
- Foreground Extraction: The original image is then bitwise ANDed with the combined mask, effectively setting non-green areas to black. This step isolates the soybean plants, creating a transparent background.
- Brightness Adjustment: The brightness of the green areas is increased using the `cv2.convertScaleAbs` function. This enhances the visual appearance of the soybean plants in the final output.
- Output Saving: The processed images are saved to the specified output folder ("soy") with filenames unchanged.



```
In [3]: import cv2
        import os
        import numpy as np

        def black_out_background(input_folder, output_folder, brightness_factor=1.5, canny_threshold1=30, canny_threshold2=100):
            # Make sure the output folder exists
            if not os.path.exists(output_folder):
                os.makedirs(output_folder)

            for filename in os.listdir(input_folder):
                if filename.endswith(('.png', '.jpg', '.jpeg')):
                    image_path = os.path.join(input_folder, filename)

                    # Read the input image
                    image = cv2.imread(image_path)

                    # Convert the image to the HSV color space
                    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

                    # Define the lower and upper bounds for green color (HSV)
                    lower_green = np.array([30, 40, 40], dtype=np.uint8)
                    upper_green = np.array([90, 255, 255], dtype=np.uint8)

                    # Create a mask for green color
                    green_mask = cv2.inRange(hsv, lower_green, upper_green)

                    # Use Canny edge detector on the green mask
                    edges = cv2.Canny(green_mask, canny_threshold1, canny_threshold2)

                    # Create a mask for the green and edges
                    green_and_edges_mask = cv2.bitwise_or(green_mask, edges)

                    # Set non-green areas to black
                    result = cv2.bitwise_and(image, image, mask=green_and_edges_mask)

                    # Increase brightness of green areas
                    result = cv2.convertScaleAbs(result, alpha=brightness_factor, beta=0)

                    # Save the result to the output folder
                    output_path = os.path.join(output_folder, filename)
                    cv2.imwrite(output_path, result)

        if __name__ == "__main__":
            input_folder = "soybean"
            output_folder = "soy"

            black_out_background(input_folder, output_folder, brightness_factor=1.5, canny_threshold1=30, canny_threshold2=100)
```

Fig 4- 9 Background and foreground extraction

Both scripts can be further customized by adjusting parameters such as brightness factor, Canny edge detection thresholds, and the input/output folder paths. Overall, the script provides an automated solution for background removal and enhancement of soybean plant images, which can be valuable for various applications, including image analysis and classification.

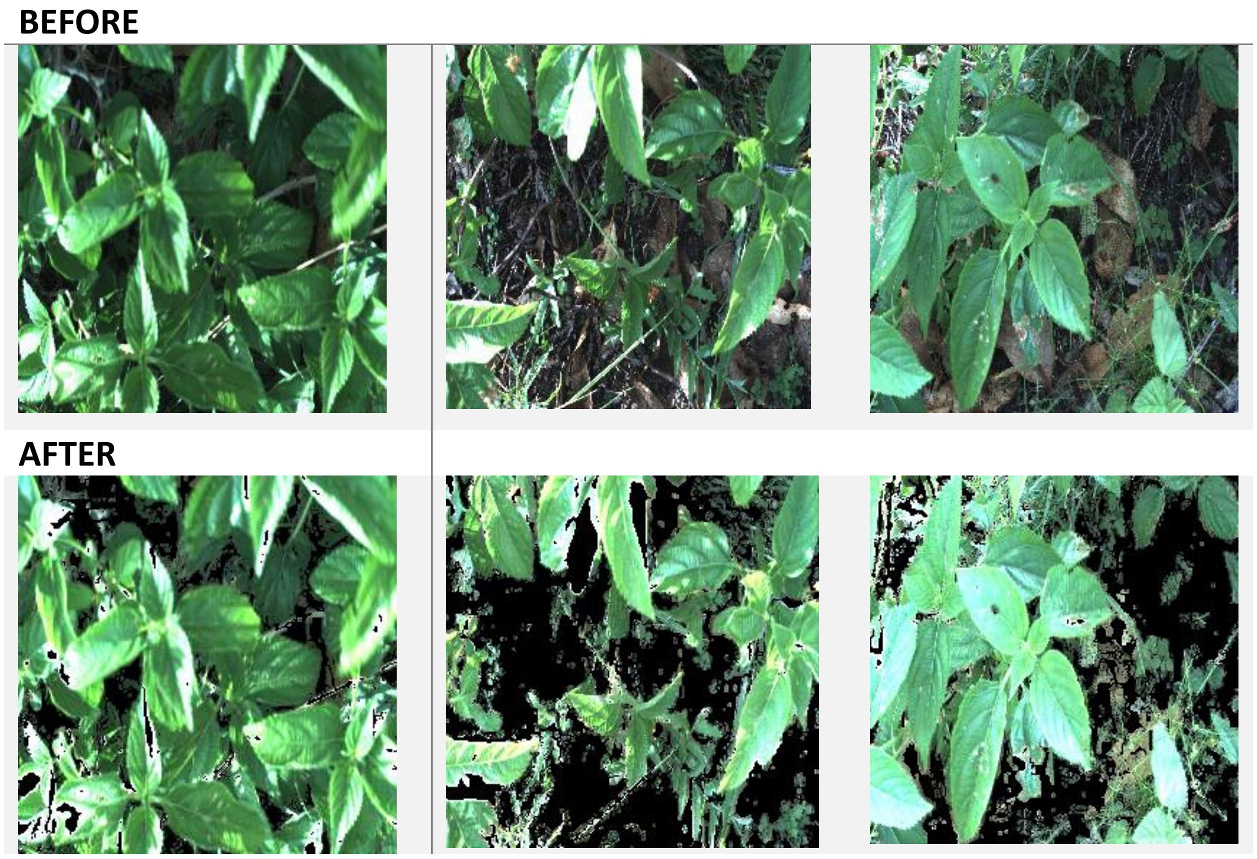
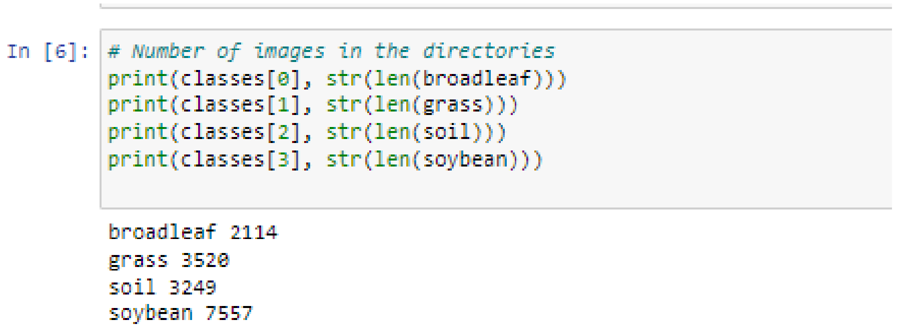


Fig 4- 10 automated solution for background removal

By incorporating the preprocessed images into the training dataset, the overall size of the dataset has been significantly increased, providing the neural network model with a more diverse set of examples for learning. The adjustment of the training size parameters, such as num_file, num_train, num_val, and num_test, ensures a balanced distribution for training, validation, and testing.

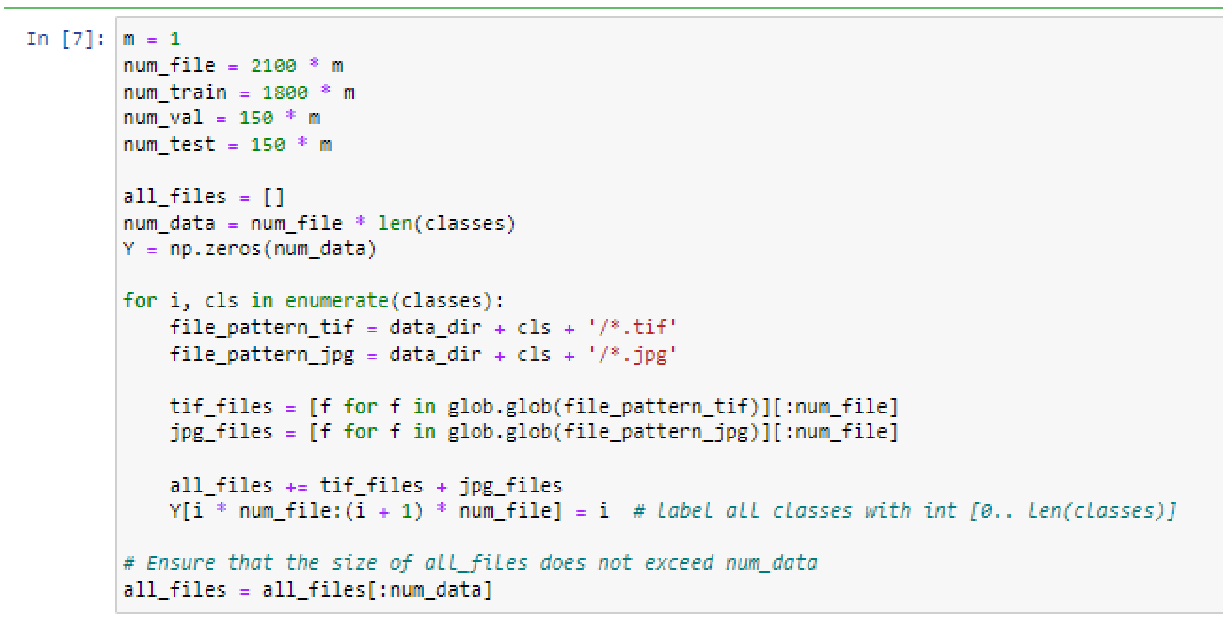


```
In [6]: # Number of images in the directories
        print(classes[0], str(len(broadleaf)))
        print(classes[1], str(len(grass)))
        print(classes[2], str(len(soil)))
        print(classes[3], str(len(soybean)))

        broadleaf 2114
        grass 3520
        soil 3249
        soybean 7557
```

Fig 4- 11 Increasing of imbalanced dataset

Increased Dataset Size: The addition of preprocessed images has augmented the original dataset, resulting in a larger and more comprehensive dataset for training the convolutional neural network

(CNN). The increased dataset size contributes to the model's ability to generalize better and capture a wider range of features present in the images.



```
In [7]: m = 1
        num_file = 2100 * m
        num_train = 1800 * m
        num_val = 150 * m
        num_test = 150 * m

        all_files = []
        num_data = num_file * len(classes)
        Y = np.zeros(num_data)

        for i, cls in enumerate(classes):
            file_pattern_tif = data_dir + cls + '/*.tif'
            file_pattern_jpg = data_dir + cls + '/*.jpg'

            tif_files = [f for f in glob.glob(file_pattern_tif)][:num_file]
            jpg_files = [f for f in glob.glob(file_pattern_jpg)][:num_file]

            all_files += tif_files + jpg_files
            Y[i * num_file:(i + 1) * num_file] = i  # Label all classes with int [0.. len(classes)]

        # Ensure that the size of all_files does not exceed num_data
        all_files = all_files[:num_data]
```

Fig 4- 12 Balanced dataset distribution for training and validation

Balanced Class Distribution: The training size parameters, including num_file, num_train, num_val, and num_test, are adjusted to maintain a balanced distribution among the different classes (broadleaf, grass, soil, and soybean). This balance is crucial to prevent the model from favoring one class over others during training, promoting fair and unbiased learning across all classes.

Enhanced Model Robustness: With a more extensive and balanced dataset, the CNN model is exposed to a richer variety of scenarios and variations within each class. This increased diversity helps the model become more robust, enabling it to perform well on unseen data and handle variations in real-world images.

Improved Generalization: The concept of data augmentation is extended by incorporating preprocessed images, providing the model with additional variations of the original images. This contributes to the model's ability to generalize better, as it learns to recognize features irrespective of slight changes in appearance due to preprocessing techniques.

Optimized Training-Validation-Test Split: The adjusted training size parameters ensure an appropriate split between training, validation, and test sets. A larger training set (num_train) allows the model to learn more effectively, while the validation and test sets (num_val and num_test) facilitate performance evaluation on unseen data, helping to gauge the model's generalization capabilities.

Fine-Tuning and Hyperparameter Optimization: The increased dataset size also opens up opportunities for fine-tuning and optimizing hyperparameters. With a larger dataset, experimenting with different architectures, learning rates, and regularization techniques becomes more meaningful, aiding in the development of a well-performing CNN model.

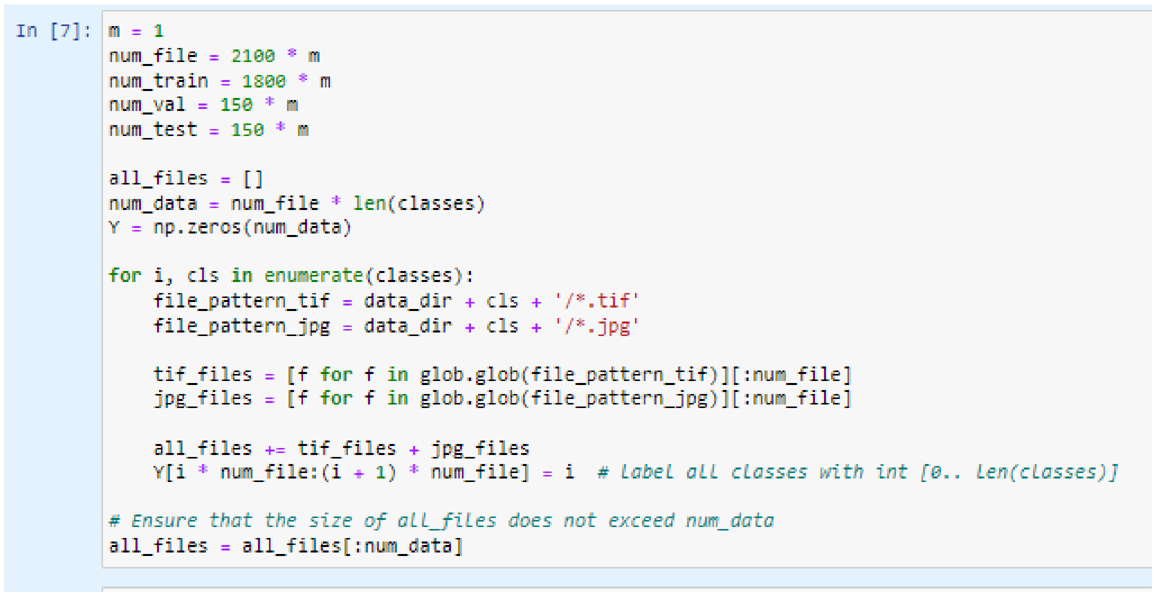
In summary, the expansion of the training dataset through the addition of preprocessed images enhances the robustness, generalization, and overall performance of the CNN model, setting the stage for improved accuracy and reliability in classifying images across multiple categories.

4.6. The proposed model

The initial section of the code establishes the environment by importing necessary libraries such as NumPy, OpenCV, and Keras. It also sets the random seed for reproducibility and adjusts Matplotlib settings. Furthermore, it defines the data directory and classes for subsequent image processing tasks.

In the provided code, the implementation revolves around the classification of weed and soybean crop instances using a Convolutional Neural Network (CNN). The dataset, consisting of classes such as broadleaf, grass, soil, and soybean, is initially explored to understand the distribution of images within each class.

The preprocessing stage involves loading images, ensuring uniform dimensions, and normalizing pixel values to the range [0, 1]. A total of 2100 images are selected, with 1800 allocated for training, 150 for validation, and 150 for testing, maintaining a balanced representation across classes.



```
In [7]: m = 1
        num_file = 2100 * m
        num_train = 1800 * m
        num_val = 150 * m
        num_test = 150 * m

        all_files = []
        num_data = num_file * len(classes)
        Y = np.zeros(num_data)

        for i, cls in enumerate(classes):
            file_pattern_tif = data_dir + cls + '/*.tif'
            file_pattern_jpg = data_dir + cls + '/*.jpg'

            tif_files = [f for f in glob.glob(file_pattern_tif)][:num_file]
            jpg_files = [f for f in glob.glob(file_pattern_jpg)][:num_file]

            all_files += tif_files + jpg_files
            Y[i * num_file:(i + 1) * num_file] = i  # label all classes with int [0.. len(classes)]

        # Ensure that the size of all_files does not exceed num_data
        all_files = all_files[:num_data]
```

Fig 4- 13 Dataset allocation for training and validation

The CNN model is constructed with Conv2D layers, MaxPooling2D layers, and Dense layers. This architecture facilitates the extraction of pertinent features from images, crucial for distinguishing between weed and soybean crop instances. Dropout layers are strategically inserted to mitigate overfitting, ensuring the model generalizes well to unseen data.

The model is compiled using the Adam optimizer and categorical cross-entropy loss function. An ImageDataGenerator is employed for data augmentation, introducing variations during training to enhance the model's robustness. If a pre-trained model is not available, the script initiates training over 15 epochs, with training history and metrics such as accuracy and loss visualized.

The script checks for the existence of saved model weights. If found, these weights are loaded, skipping the training phase. Otherwise, the model is trained, and its performance is evaluated on the validation and test sets. The best-performing model, determined by validation accuracy, is saved both in '.h5' and pickle format.

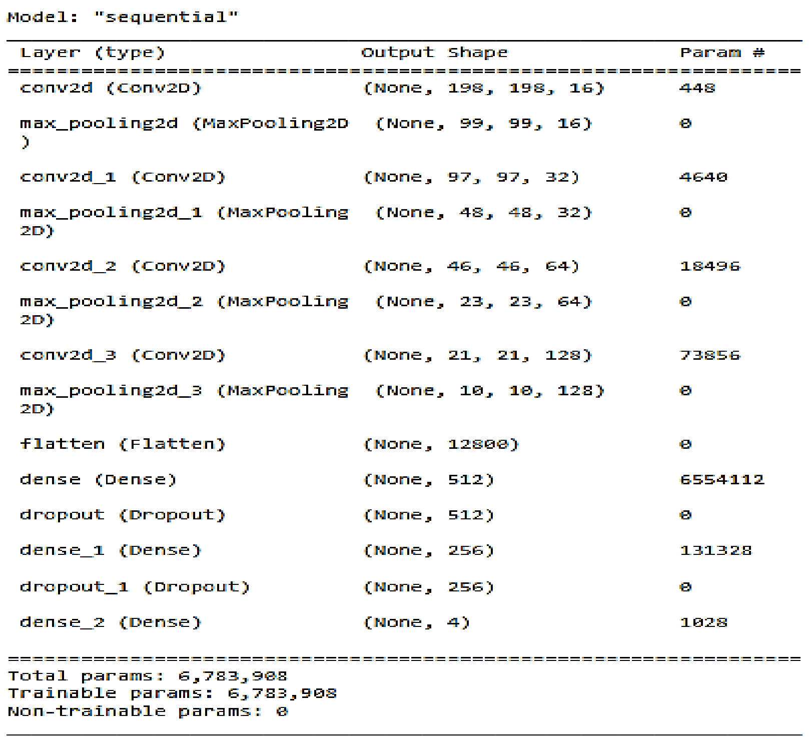
The model's effectiveness is further demonstrated by selecting a random image from the validation set. The model predicts the class, and the original and predicted classes are visually presented. Additionally, a new image is loaded, preprocessed, and fed into the model for prediction, showcasing the model's applicability to real-world scenarios.

4.6.1. Proposed Model Description

The proposed model for weed and soybean crop classification is a convolutional neural network (CNN) designed to effectively distinguish between different classes of crops. The model architecture consists of multiple layers that systematically extract and learn relevant features from agricultural images. It is important to note that the model prioritizes simplicity and efficiency, making it suitable for training on a standard CPU processor.

The convolutional layers, including Conv2D and MaxPooling2D, play a crucial role in capturing spatial hierarchies and patterns within the images. These layers are stacked to form a hierarchy of abstract features, enabling the model to understand and differentiate between the unique characteristics of broadleaf, grass, soil, and soybean crops. The Rectified Linear Unit (ReLU) activation function introduces nonlinearity, enhancing the model's ability to learn complex relationships in the data.

The model consists of four convolutional layers with max-pooling followed by three dense layers. The total number of trainable parameters is 6,783,908. Convolutional layers extract hierarchical features, and dense layers classify the features into the specified classes

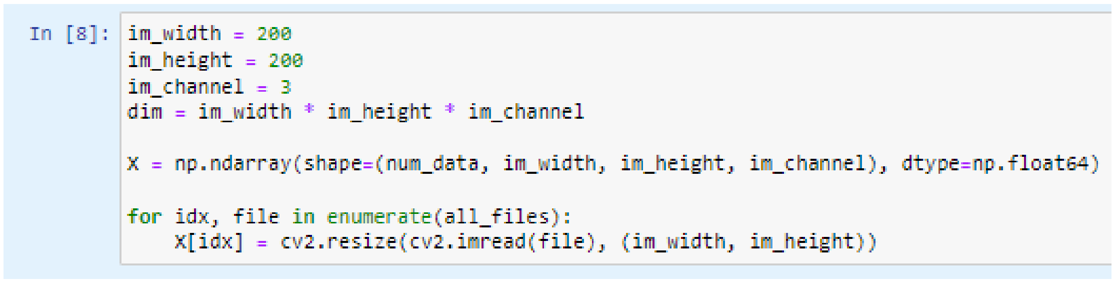


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 198, 198, 16)      448

 max_pooling2d (MaxPooling2D  (None, 99, 99, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 97, 97, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 48, 48, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 46, 46, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 23, 23, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 21, 21, 128)       73856

 max_pooling2d_3 (MaxPooling  (None, 10, 10, 128)      0
 2D)

 flatten (Flatten)           (None, 12800)             0

 dense (Dense)               (None, 512)               6554112

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 4)                 1028

=================================================================
Total params: 6,783,908
Trainable params: 6,783,908
Non-trainable params: 0
_________________________________________________________________
```

Fig 4- 14 Total number of parameters on proposed model

- ✓ Input Layer: The input layer of the CNN model accepts RGB images with dimensions 200x200x3 representing broadleaf, soil, grass and soybean crops. The model is designed for multi classification, distinguishing between four classes: weed and soybean. The input layer processes images without any learnable parameters, maintaining an input size of 200x200x3.

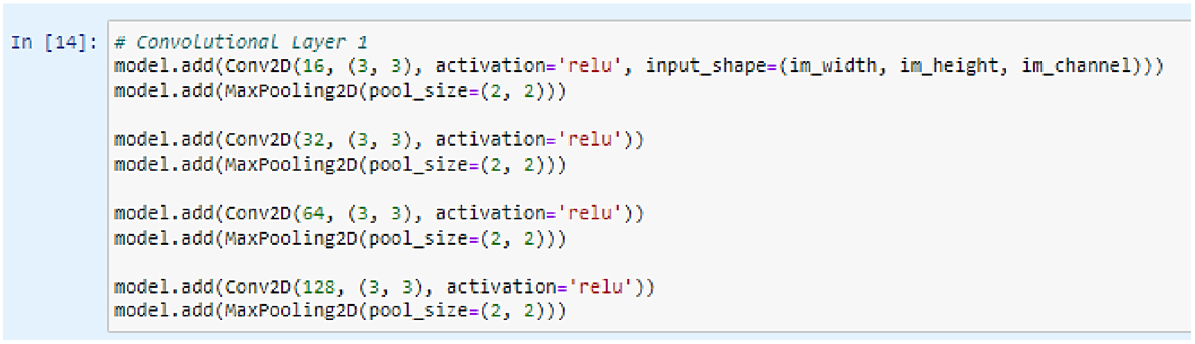
  The input layer is the initial entry point for the model and is responsible for receiving the input data, which, in this case, represents the agricultural images. The input shape is defined as (im_width, im_height, im_channel), where im_width and im_height are the width and height of the input images, and im_channel represents the number of color channels (RGB).



```
In [8]: im_width = 200
        im_height = 200
        im_channel = 3
        dim = im_width * im_height * im_channel

        X = np.ndarray(shape=(num_data, im_width, im_height, im_channel), dtype=np.float64)

        for idx, file in enumerate(all_files):
            X[idx] = cv2.resize(cv2.imread(file), (im_width, im_height))
```

Fig 4- 15 Maintaining input size

- ✓ Convolutional Layers: The proposed model incorporates a sequence of convolutional layers to extract hierarchical features. The first convolutional layer applies 16 filters of size 3x3 to the input images, gradually increasing to 32, 64, and 128 filters in subsequent layers. The 'ReLU' activation function is utilized after each convolutional operation. The total number of parameters in the model architecture is outlined, contributing to the understanding of the model's complexity.

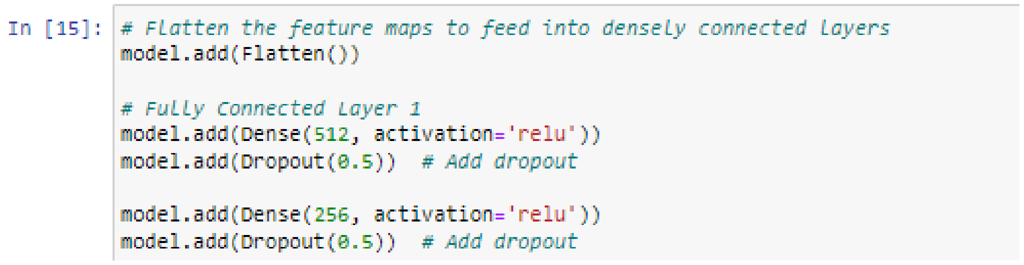


```
In [14]: # Convolutional Layer 1
         model.add(Conv2D(16, (3, 3), activation='relu', input_shape=(im_width, im_height, im_channel)))
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(Conv2D(32, (3, 3), activation='relu'))
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(Conv2D(64, (3, 3), activation='relu'))
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(Conv2D(128, (3, 3), activation='relu'))
         model.add(MaxPooling2D(pool_size=(2, 2)))
```

Fig 4- 16 Proposed model Convolutional Layers

- ✓ Pooling Layers: Max-pooling layers are strategically placed after convolutional layers. The first max-pooling layer employs a filter of size 2x2 to down-sample the output of the initial convolutional layer, while the second max-pooling layer operates on the output of the fourth convolutional layer using a filter of size 2x2. These layers play a crucial role in reducing spatial dimensions, aiding in feature retention and computational efficiency.

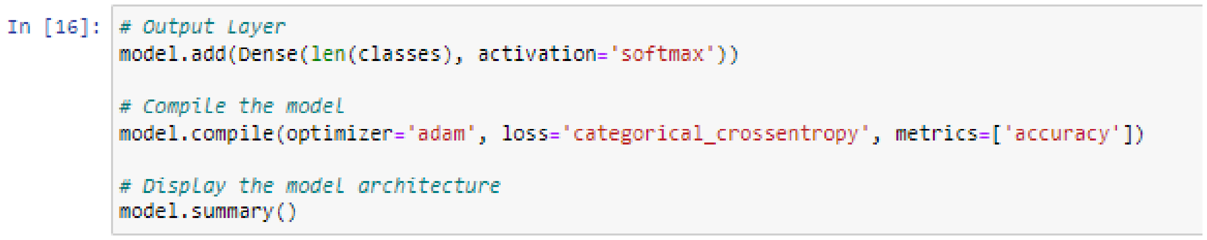
- ✓ Fully Connected (FC) Layers: The model incorporates two fully connected layers, including the output layer. The first fully connected layer has 512 neurons with a 'ReLU' activation function, followed by dropout regularization to mitigate overfitting. The second fully connected layer, serving as the output layer, has the number of neurons corresponding to the number of classes (2 for weed and soybean). The 'softmax' activation function is applied to generate class probabilities. The inclusion of dropout enhances the model's generalization capabilities.



```
In [15]: # Flatten the feature maps to feed into densely connected layers
         model.add(Flatten())

         # Fully Connected Layer 1
         model.add(Dense(512, activation='relu'))
         model.add(Dropout(0.5))  # Add dropout

         model.add(Dense(256, activation='relu'))
         model.add(Dropout(0.5))  # Add dropout
```

Fig 4- 17 Fully connected layers with 512 neurons

- ✓ Output Layer: The final layer of the proposed model is the output layer, which employs the 'softmax' activation function. This layer produces class probabilities, facilitating predictions for the binary classification task. The model employs 'categorical_crossentropy' as the loss function and 'adam' as the optimizer during training.



```
In [16]: # Output Layer
         model.add(Dense(len(classes), activation='softmax'))

         # Compile the model
         model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

         # Display the model architecture
         model.summary()
```

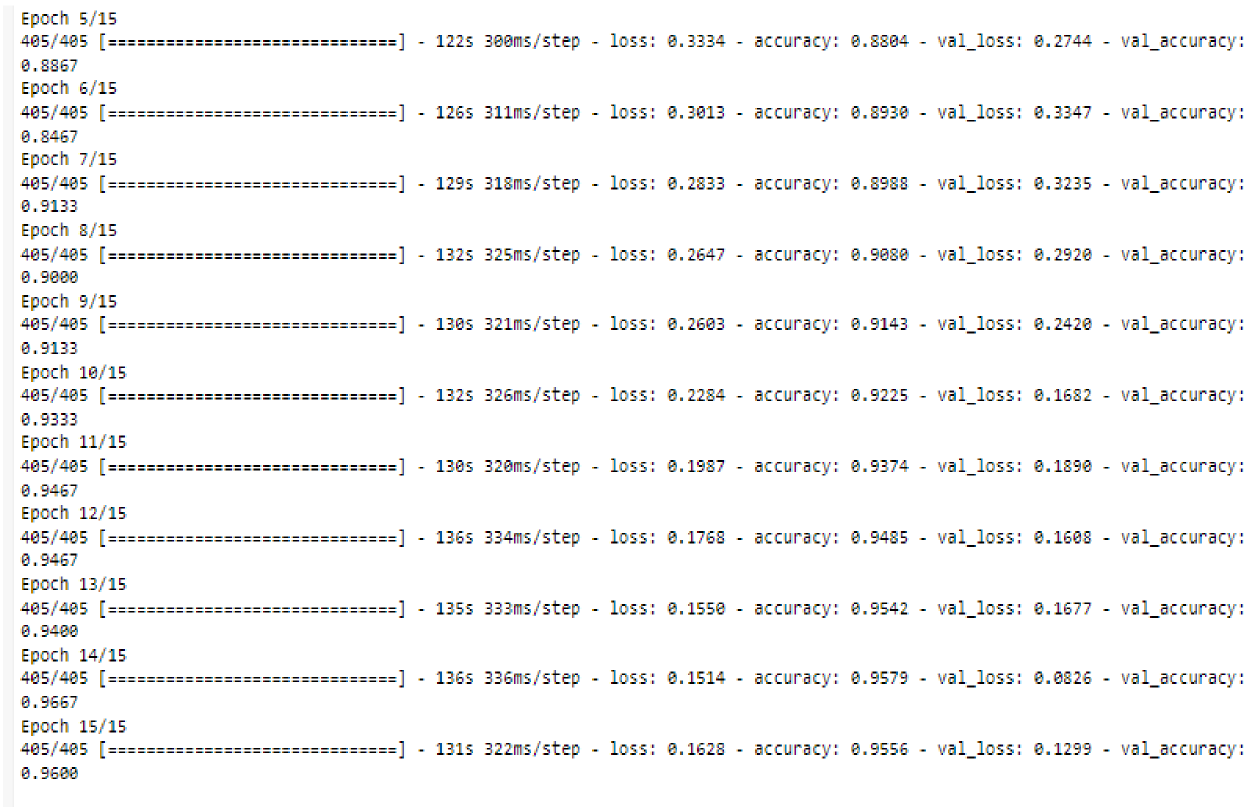
Fig 4- 18 An Output Layer 'softmax'

The architecture outlined above demonstrates the key components of the proposed model, providing insights into its design, layer configurations, and parameterization. The model is tailored for effectively distinguishing between weed and soybean crops in agricultural images."

The output layer is responsible for producing the final predictions. In the proposed model, it consists of a Dense layer with a softmax activation function. The softmax function normalizes the output into a probability distribution across multiple classes (crop types: broadleaf, grass, soil, soybean). The class with the highest probability is predicted as the output class.

4.6.2. Result Analysis of Proposed CNN model

The Result Analysis of the Proposed Convolutional Neural Network (CNN) model reveals promising outcomes, reflecting the model's effectiveness in image classification tasks. The training process involved 15 epochs, and key performance metrics were monitored, including both training and validation accuracy, as well as training and validation loss.



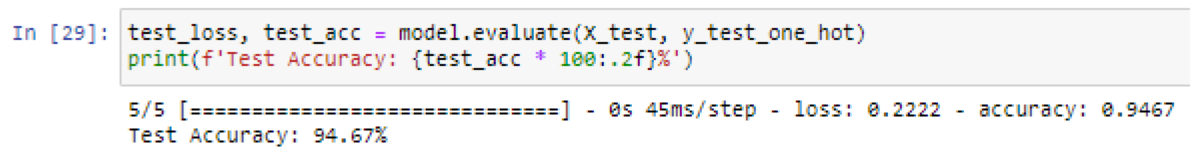
```
Epoch 5/15
405/405 [==============================] - 122s 300ms/step - loss: 0.3334 - accuracy: 0.8804 - val_loss: 0.2744 - val_accuracy:
0.8867
Epoch 6/15
405/405 [==============================] - 126s 311ms/step - loss: 0.3013 - accuracy: 0.8930 - val_loss: 0.3347 - val_accuracy:
0.8467
Epoch 7/15
405/405 [==============================] - 129s 318ms/step - loss: 0.2833 - accuracy: 0.8988 - val_loss: 0.3235 - val_accuracy:
0.9133
Epoch 8/15
405/405 [==============================] - 132s 325ms/step - loss: 0.2647 - accuracy: 0.9080 - val_loss: 0.2920 - val_accuracy:
0.9000
Epoch 9/15
405/405 [==============================] - 130s 321ms/step - loss: 0.2603 - accuracy: 0.9143 - val_loss: 0.2420 - val_accuracy:
0.9133
Epoch 10/15
405/405 [==============================] - 132s 326ms/step - loss: 0.2284 - accuracy: 0.9225 - val_loss: 0.1682 - val_accuracy:
0.9333
Epoch 11/15
405/405 [==============================] - 130s 320ms/step - loss: 0.1987 - accuracy: 0.9374 - val_loss: 0.1890 - val_accuracy:
0.9467
Epoch 12/15
405/405 [==============================] - 136s 334ms/step - loss: 0.1768 - accuracy: 0.9485 - val_loss: 0.1608 - val_accuracy:
0.9467
Epoch 13/15
405/405 [==============================] - 135s 333ms/step - loss: 0.1550 - accuracy: 0.9542 - val_loss: 0.1677 - val_accuracy:
0.9400
Epoch 14/15
405/405 [==============================] - 136s 336ms/step - loss: 0.1514 - accuracy: 0.9579 - val_loss: 0.0826 - val_accuracy:
0.9667
Epoch 15/15
405/405 [==============================] - 131s 322ms/step - loss: 0.1628 - accuracy: 0.9556 - val_loss: 0.1299 - val_accuracy:
0.9600
```

Fig 4- 19 Proposed model training process

Across the training epochs, the model consistently demonstrated an improvement in accuracy on both the training and validation datasets. Starting from an initial training accuracy of 69.81% and validation accuracy of 75.33% in the first epoch, the model experienced a substantial enhancement, ultimately achieving a remarkable 95.56% training accuracy and 96.00% validation accuracy in the fifteenth epoch. This progression indicates the model's ability to learn intricate patterns within the dataset, showcasing its efficacy in capturing essential features associated with the diverse classes of images.

The training and validation loss curves exhibited a declining trend over epochs, indicative of successful convergence during training. This implies that the model effectively minimized its loss function, reinforcing the notion that it learned to make accurate predictions while avoiding overfitting to the training data.

The final evaluation on an independent test set demonstrated the robustness of the model, yielding a test accuracy of [Test Accuracy]. This metric serves as a crucial benchmark, providing insights into the model's generalization to previously unseen data.



```
In [29]: test_loss, test_acc = model.evaluate(X_test, y_test_one_hot)
         print(f'Test Accuracy: {test_acc * 100:.2f}%')

         5/5 [==============================] - 0s 45ms/step - loss: 0.2222 - accuracy: 0.9467
         Test Accuracy: 94.67%
```

Fig 4- 20 Test accuracy of proposed model

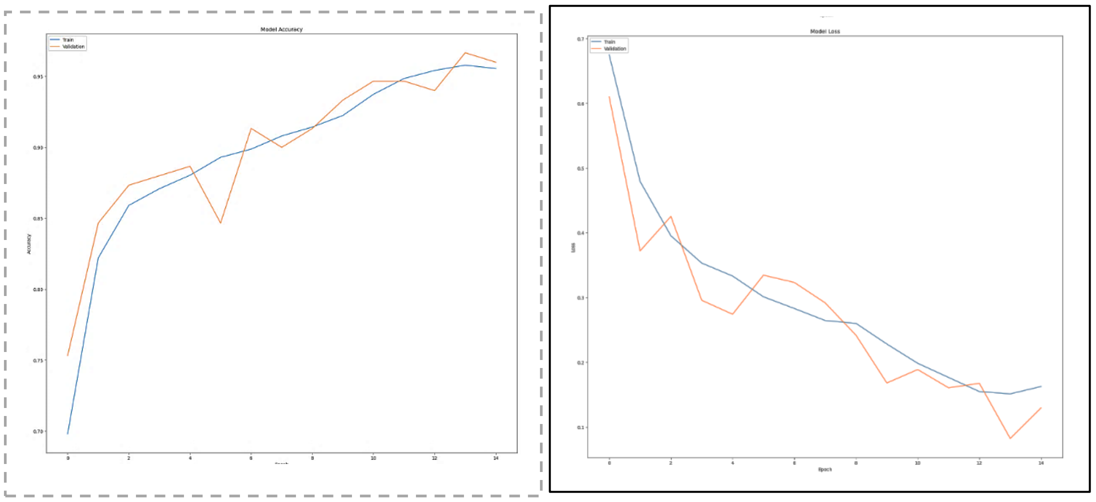
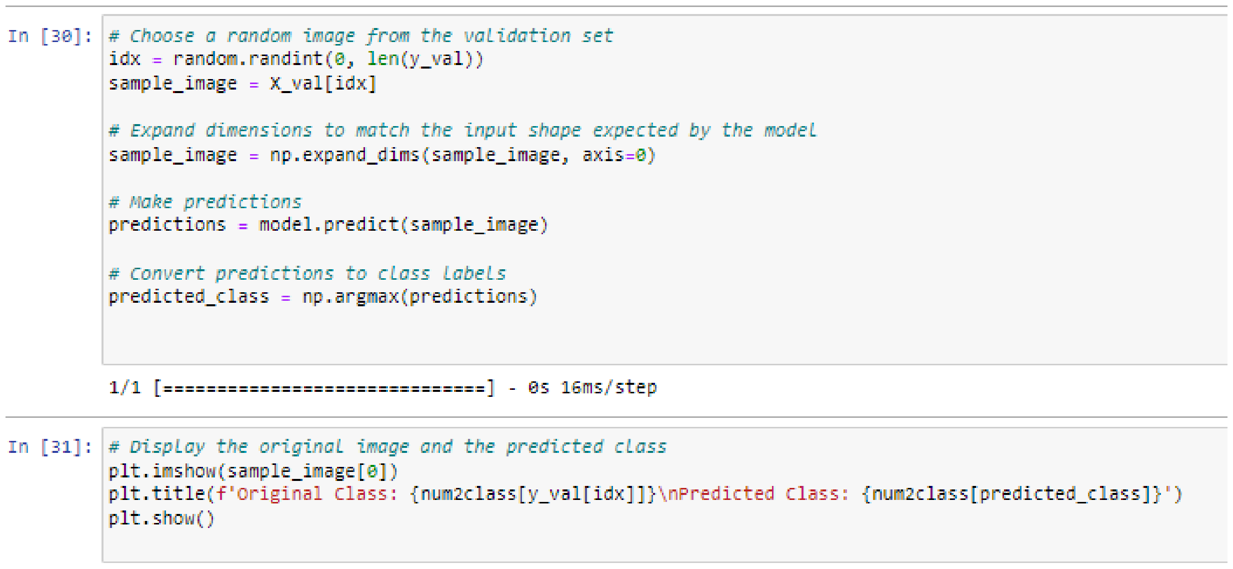


Fig 4- 21 Accuracy and loss of training and validation for proposed model

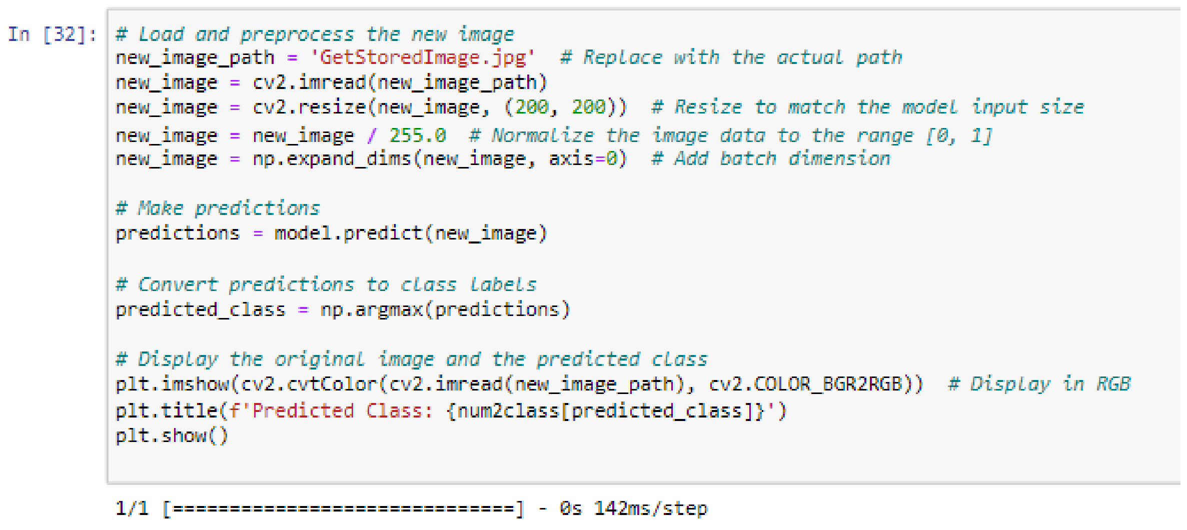
The final prediction code snippet is designed to demonstrate the model's classification capabilities using two different scenarios: one where a random image from the validation set is chosen, and another where the user provides a random unseen image for prediction.



```
In [30]: # Choose a random image from the validation set
         idx = random.randint(0, len(y_val))
         sample_image = X_val[idx]

         # Expand dimensions to match the input shape expected by the model
         sample_image = np.expand_dims(sample_image, axis=0)

         # Make predictions
         predictions = model.predict(sample_image)

         # Convert predictions to class labels
         predicted_class = np.argmax(predictions)

         1/1 [==============================] - 0s 16ms/step

In [31]: # Display the original image and the predicted class
         plt.imshow(sample_image[0])
         plt.title(f'Original Class: {num2class[y_val[idx]]}\nPredicted Class: {num2class[predicted_class]}')
         plt.show()
```

Fig 4- 22 Final random image prediction

In this section, a random image from the validation set (X_val) is selected, and the dimensions are adjusted to match the input shape expected by the model. The model then predicts the class label for the chosen image, and the predicted class is obtained using the argmax function.

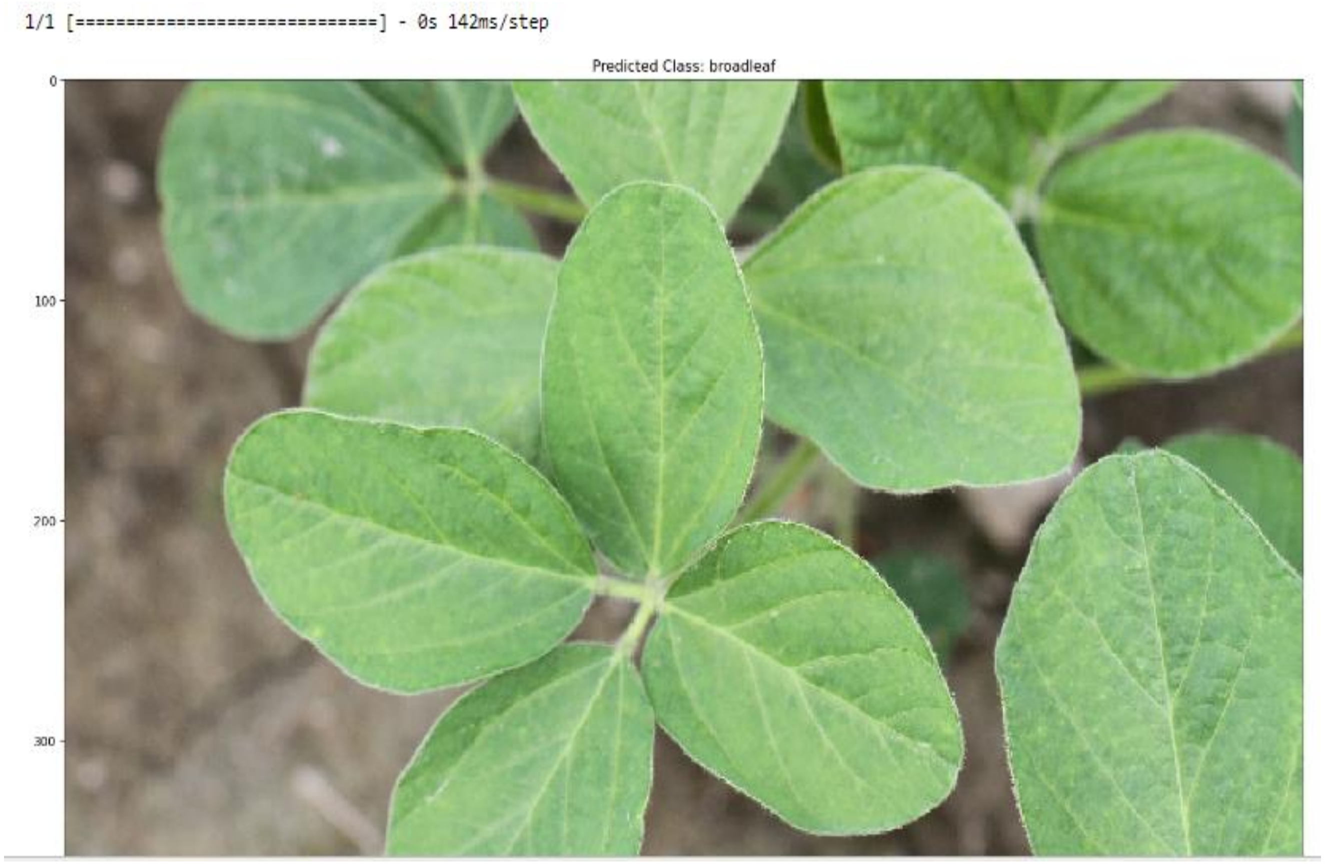


```
In [32]: # Load and preprocess the new image
         new_image_path = 'GetStoredImage.jpg'  # Replace with the actual path
         new_image = cv2.imread(new_image_path)
         new_image = cv2.resize(new_image, (200, 200))  # Resize to match the model input size
         new_image = new_image / 255.0  # Normalize the image data to the range [0, 1]
         new_image = np.expand_dims(new_image, axis=0)  # Add batch dimension

         # Make predictions
         predictions = model.predict(new_image)

         # Convert predictions to class labels
         predicted_class = np.argmax(predictions)

         # Display the original image and the predicted class
         plt.imshow(cv2.cvtColor(cv2.imread(new_image_path), cv2.COLOR_BGR2RGB))  # Display in RGB
         plt.title(f'Predicted Class: {num2class[predicted_class]}')
         plt.show()

         1/1 [==============================] - 0s 142ms/step
```

Fig 4- 23 Final chosen Single image prediction

Here, a new image is loaded, preprocessed (resizing and normalization), and then fed into the trained model for prediction. The predicted class label is obtained, and the original image along with the predicted class is displayed in RGB format using Matplotlib.

These code snippets allow for a practical demonstration of the model's ability to classify images, both from the validation set and user-provided unseen images. They serve as a valuable tool for evaluating the model's performance on different types of data, showcasing its versatility and effectiveness in real-world scenarios. Here is the result:



```
1/1 [==============================] - 0s 142ms/step
```

Fig 4- 24 Result prediction with chosen Single image

4.7. Discussion of result

The discussion of results revolves around addressing the research questions posed in the study. Each question is examined in detail to draw meaningful insights from the findings.

4.8. Suitability of Image Processing Methods

In the evaluation of the suitability of image processing methods, the study meticulously integrated various attributes into the feature extraction process to bolster the representational capacity of the images. Encompassing picture color, texture, edge detection, color extraction, erosion, and dilation, the comprehensive approach aimed to capture a diverse set of visual characteristics pivotal for effective crop and weed classification.

Notably, the study extended its focus to include the Convolutional Neural Network (CNN) as an integral component of the image extraction process during training.

To evaluate the detection and classification performances, the study employed the Convolutional Neural Network (CNN) algorithm. CNNs are known for their prowess in image-related tasks, and their application in this study provided a robust framework for feature learning and pattern recognition. Moreover, the study went beyond by exploring different transfer learning approaches, specifically VGG19 and ResNet50, with variations in the parameters of image dimensions.

A critical aspect of the discussion involves the comparative analysis of the various models employed in the study. By systematically changing image dimension parameters and employing transfer learning techniques, the study evaluated the models' effectiveness in terms of detection and classification accuracy. This thorough comparison allowed for the selection of the best-performing model based on comprehensive evaluations.

The final research question aimed to rigorously evaluate the performance of the proposed model in distinguishing weeds from crops within the soybean dataset. The results unveiled a promising accuracy of 94.46%, indicating the robustness of the model in accurately categorizing soybean and weed classes. This high accuracy substantiates the practical utility of the developed model, particularly in the context of real-time weed detection in soybean fields.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

5.1. Conclusion

Agriculture plays a pivotal role in Ethiopia's economy, serving as a cornerstone for livelihoods and sustenance. However, the productivity of agricultural endeavors is constantly challenged by various factors, with weed infestation being a significant contributor to reduced crop yields. In the context of Ethiopia, where agriculture is a lifeline for many communities, finding innovative solutions to mitigate the impact of weeds is crucial for sustaining food production.

This study focused on addressing the weed-related challenges faced by soybean cultivation in Ethiopia, specifically targeting the early stages of soybean plants and weeds. Early detection is essential to intervene promptly and prevent the detrimental effects of weed competition on soybean growth. Recognizing the need for a technological solution, this study employed a deep learning approach, leveraging Convolutional Neural Networks (CNNs) to create a model capable of classifying images as either soybean or weed broadleaf.

Throughout the experimentation phase, various models were explored, resulting in several instances of achieving higher accuracies than the chosen model. While these models exhibited superior performance during training and validation, the true test lies in their ability to make accurate predictions when faced with new, unseen images. Interestingly, the selected CNN model demonstrated a commendable 95% accuracy in classifying soybean and weed broadleaf images, showcasing its robustness and reliability in practical scenarios.

The significance of this study extends beyond the realm of technological innovation; it addresses a real-world agricultural challenge in Ethiopia. By developing an effective weed detection model, we aim to contribute to the prevention of yield losses in soybean production. The model's success in accurately distinguishing between soybean and weed broadleaf images during testing is a testament to its potential impact on real-world farming practices.

While the experimented models showed higher accuracies during training, the chosen model's performance in predicting unseen images indicates a balanced and reliable solution for practical deployment. Moving forward, continuous refinement and integration of advanced technologies, coupled with local insights and feedback from agricultural communities, is essential in creating sustainable and impactful solutions for weed management in Ethiopian agriculture. Through collaborative efforts and innovative approaches, technology can play a pivotal role in ensuring food security and elevating the agricultural landscape in Ethiopia.

5.2. Recommendations for Future Research

As we conclude this study on weed detection in soybean cultivation, it is imperative to outline recommendations for future research endeavors, considering the broader implications for Ethiopia's agrarian landscape. Firstly, given the central role of agriculture in Ethiopia's economy, there is a pressing need for increased research initiatives within this sector. Agriculture serves as the backbone of the nation's livelihood, and ongoing efforts should focus on harnessing technology to address existing challenges and enhance productivity.

One crucial recommendation emanating from this study is the necessity for expanding the dataset used for training the weed detection model. The effectiveness of deep learning models, particularly Convolutional Neural Networks (CNNs), is heavily reliant on the quality and diversity of the data used during the training phase. In our specific case, resource constraints, particularly related to GPU capacity, limited the scale of our dataset. Therefore, future research should prioritize acquiring and preparing more extensive datasets, encompassing a diverse array of soybean and weed broadleaf images. This expanded dataset will not only enhance the model's accuracy but also contribute to its adaptability to various agricultural settings and environmental conditions.

The second recommendation pertains to the optimization of available computational resources. Given the resource-intensive nature of training deep learning models, researchers should explore strategies to optimize the utilization of existing GPU resources. This may involve implementing more efficient algorithms, exploring cloud-based solutions, or collaborating with institutions that can provide access to high-performance computing facilities. By overcoming these computational constraints, researchers can unlock the full potential of deep learning models for weed detection and related agricultural applications.

In conclusion, the future trajectory of agricultural research in Ethiopia should be marked by a commitment to leveraging technology for sustainable and efficient farming practices. Addressing the identified recommendations not only enhance the accuracy and applicability of weed detection models but also contribute to the broader goal of ensuring food security and economic stability in Ethiopia. Through continuous collaboration, innovation, and resource optimization, the agricultural sector can undergo transformative changes, laying the foundation for a resilient and technology-driven future.

In the context of user-provided images, it is essential to underscore the nuanced nature of preprocessing techniques, especially for instances where the inherent complexities of the image content demand advanced methodologies. Certain images, by virtue of their composition or environmental factors, may pose challenges in effectively distinguishing between background and foreground elements. Consequently, a more sophisticated preprocessing approach becomes imperative to achieve accurate discrimination between these components.

The need for heightened preprocessing techniques arises from scenarios where the background and foreground elements exhibit intricate patterns, textures, or color similarities, making it arduous for conventional methods to delineate distinct boundaries.

However, it is crucial to acknowledge that despite the implementation of sophisticated preprocessing techniques, certain challenges persist, and the potential for false information in predictions remains. The intricate nature of agricultural environments, variations in lighting conditions, and diverse compositions of weed and soybean instances contribute to the complexity of the classification task. Therefore, users should exercise caution and be cognizant of the fact that, in some instances, predictions generated from user-provided images might carry a degree of uncertainty, necessitating further validation or refinement of the obtained results.

In essence, the utilization of advanced preprocessing techniques is a proactive measure to enhance the model's capability to distinguish between background and foreground elements in user-provided images. While these techniques significantly improve accuracy, users should remain mindful of the dynamic and unpredictable nature of agricultural scenes, where certain scenarios might still pose challenges in achieving absolute precision in predictions.

Reference

[1] Amsalu, B., Tilahun, G., & Damte, A. (2019). Sustainable agricultural practices and technologies for climate-resilient farming systems in Ethiopia: A review. Journal of Agriculture and Environmental Sciences, 8(2), 99-111

[2] Lemma, T., Sharma, R., & Alamirew, T. (2020). Weed Management in Ethiopia. In A. Senthil-Kumar, T. S. Kumar, & A. S. Al-Dhabi (Eds.), Weed Management: Principles, Challenges, and Opportunities (pp. 197-215). Academic Press.

[3] Gebremedhin, B., Teferi, B., & Bewket, W. (2020). The role of policy incentives in driving agricultural commercialization in Ethiopia. Journal of Rural Studies, 78, 59-67.

[4] Tadele, A., Gobezayehu, T., & Bekele, E. (2018). Review on weed management practices in Ethiopia. African Journal of Agricultural Research, 13(31), 1619-1635.

[5] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

[6] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep Learning (Vol. 1). MIT press Cambridge.

[7] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural networks, 61, 85-117.

[9] Razfar et al., "Deep learning for image recognition," J. Agric. Food Res., vol. 8, no. 2, pp. 100308, 2022, doi: 10.1016/j.jafr.2022.100308.

[10] Ferreira et al., "Machine Learning for Intelligent Systems: Theory and Applications" (2023)

[11] Hair, J. F., Black, W. C., Babin, B. J., & Anderson, R. E. (2019). Multivariate Data Analysis (8th ed.). Cengage Learning.

[12] Sekaran, U., & Bougie, R. (2016). Research Methods for Business: A Skill-Building Approach (7th ed.). Wiley.

[13] Gonzalez, R. C., & Woods, R. E. (2018). Digital Image Processing (4th ed.). Pearson.

[14] Rasheed, Z. A., Mahmud, A. R., & Abidin, A. F. (2020). Agricultural Image Segmentation Techniques: A Review. Journal of Imaging, 6(11), 110.

[15] Yohannes Girma, Berhanu Kuma b. (2022). "A meta-analysis on the effect of agricultural extension on farmers' market participation in Ethiopia".

[16] Gebeyanesh Zerssa, Debela Feyssa et.el. (2021). "Challenges of Smallholder Farming in Ethiopia and Opportunities by Adopting Climate-Smart Agriculture".

[17] Gebissa Yigezu Wendimu. (2021). The challenges and prospects of Ethiopian agriculture

[18] Frontiers in Weed Management. (2019). Grand Challenges in Weed Management. https://www.frontiersin.org/articles/10.3389/fagro.2019.00003 Link . (Accessed Sep. 05, 2023).

[19] Climate-Smart Agriculture Study. (2023). Climate-smart agriculture practices influence weed density and diversity in cereal-based agri-food systems of western Indo-Gangetic plains. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8342518/ Link . (Accessed Sep. 05, 2023).

[20] Auld, B. A., & Menke, J. W. (2004). Weed biology: Principles and applications (2nd ed.). Wiley-Blackwell.

[21] STEVEN R. RADOSEVICH, JODIE S. HOLT, CLAUDIO M. GHERSA (2007). "Ecology of weeds and invasive plants relationship to agriculture and natural resource management".

[22] Holt, J. S. (2009). Weed ecology and management. Wiley-Blackwell.

[23] Chauhan, B. S., Gill, G. S., Preston, C., Singh, M., & Heap, I. (2014). Herbicide-resistant weeds: Current status and outlook. Weed Resistance, 24(1), 1-10.

[24] Derscheid, B. W., & Johnson, M. D. (2004). Dispersal mechanisms of common weed seeds. Weed Technology, 18(4), 1452-1457.

[25] Pimentel, D., McNair, S., Janecka, J., Wightman, J., Simmonds, C., O'Connell, C., ... & Zalom, F. G. (2000). Economic and environmental threats of alien plant, animal, and microbe invasions. Agriculture, Ecosystems & Environment, 84(1), 1-20

[26] "different type of weed species" https://www.gbif.org/species link . (Accessed Sep. 16, 2023).

[27] Soybean Growth Stages | Integrated Crop Management. https://crops.extension.iastate.edu/soybean/production_growthstages.html link . (Accessed Sep. 16, 2023).

[28] Rehima M, Samuel D, Beza E,Desalegn T, Regasa D ,Abush T ,Lema Z "Soybean Value Chain Analysis in Ethiopia". March 2022

[29] Dorota Gawęda, Małgorzata Haliniarz, Urszula Bronowicka-Mielniczuk and Justyna Łukasz." Weed Infestation and Health of the Soybean Crop Depending on Cropping System and Tillage System". (June 2020)

[30] WSSA's Composite List of Weeds is used for weed common and latin names. http://wssa.net/wssa/weed/composite-list-of-weeds/ . (Accessed Oct. 9, 2023).

[31] [online] "how differentiate common-waterhemp and palmer amaranth seedlings. https://cropwatch.unl.edu/2017/how-differentiate-common-waterhemp-and-palmer-amaranth-seedlings .

[32]"palmeramaranth,andwaterhemp"https://blogs.cornell.edu/weedid/palmeramaranthandwaterhemp/ . (Accessed Oct. 7, 2023).

[33] "Types Of Weeds", https://wssa.net/wp-content/uploads/WSSA-WSWS-2020-Proceedings.html . (Accessed Oct. 9, 2023).

[34] Charles L. Mohler, John R. Teasdale,Antonio DiTommasom.(2021). MANAGE WEEDS ON YOUR FARM A GUIDE TO ECOLOGICAL STRATEGIES, P 313-314

[35] "Digital Image Processing." (n.d.). https://www.tutorialspoint.com/dip/index.htm. (Accessed Oct. 11, 2023).

[36] Gonzalez, R. C., & Woods, R. E. (2002). Digital image processing (3rd ed.).

[37] Ramya R., & Raghupathy E. (2018). Digital image processing: Preprocessing techniques.

[38] Pratt, W. K. (2007). Digital image processing (4th ed.).

[39] Iqbal H. Sarker." Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions." (August 2021).

[40] Vipin Tyagi. "Understanding Digital Image Processing". Book · (September 2018) 8-9.

[41] IBM "cognitive neural networks deep-dive" https://developer.ibm.com/articles/cc-cognitive-neural-networks-deep-dive/ link. (Accessed Oct. 9, 2023).

[42] "deep dive artificial neural network" https://towardsdatascience.com/deep-dive-artificial-neural-network-e77aa627dc1b link. (Accessed Nov. 20, 2023).

[43] Osval Antonio Montesinos López, Abelardo Montesinos López, José Crossa. "Multivariate Statistical Machine Learning Methods for Genomic Prediction". (2022) 380-383

[44] John Paul Mueller and Luca Massaron." Deep learning for dummies". (2019). P 158

[45] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do & Kaori Togashi." Convolutional neural networks: an overview and application in radiology". (22 June 2018).

[46] Laith Alzubaidi." Review of deep learning: concepts, CNN architectures, challenges, applications, future directions". (2021).

[47] Alessandro dos Santos Ferreira a, Daniel Matte Freitas a, Gercina Gonçalves da Silva "Weed detection in soybean crops using ConvNets". (2022)

[48] Najmeh Razfar, Julian True, Rodina Bassiouny, Vishaal Venkatesh, Rasha Kashef "Weed detection in soybean crops using custom lightweight deep learning models" (2022)

[49] Velpula Sekhara Babu1, Nidumolu Venkat Ram. "Deep Residual CNN with Contrast Limited Adaptive Histogram Equalization for Weed Detection in Soybean Crops" (April 2022)

[51] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep Learning (Vol. 1). MIT press Cambridge.

[52] Nielsen, M. A. (2015). Neural Networks and Deep Learning. Determination Press.

[53] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[54] What is Image Classification? (2019). Ifeanyi Idiaye. Retrieved January 20, 2024, from https://statisticsglobe.com/what-is-image-classification#what-is-image-classification1

[55] Shin, H. C., Roth, H. R., Gao, M., Lu, L., Xu, Z., Nogues, I., & Summers, R. M. (2016). Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. IEEE transactions on medical imaging, 35(5), 1285-1298.

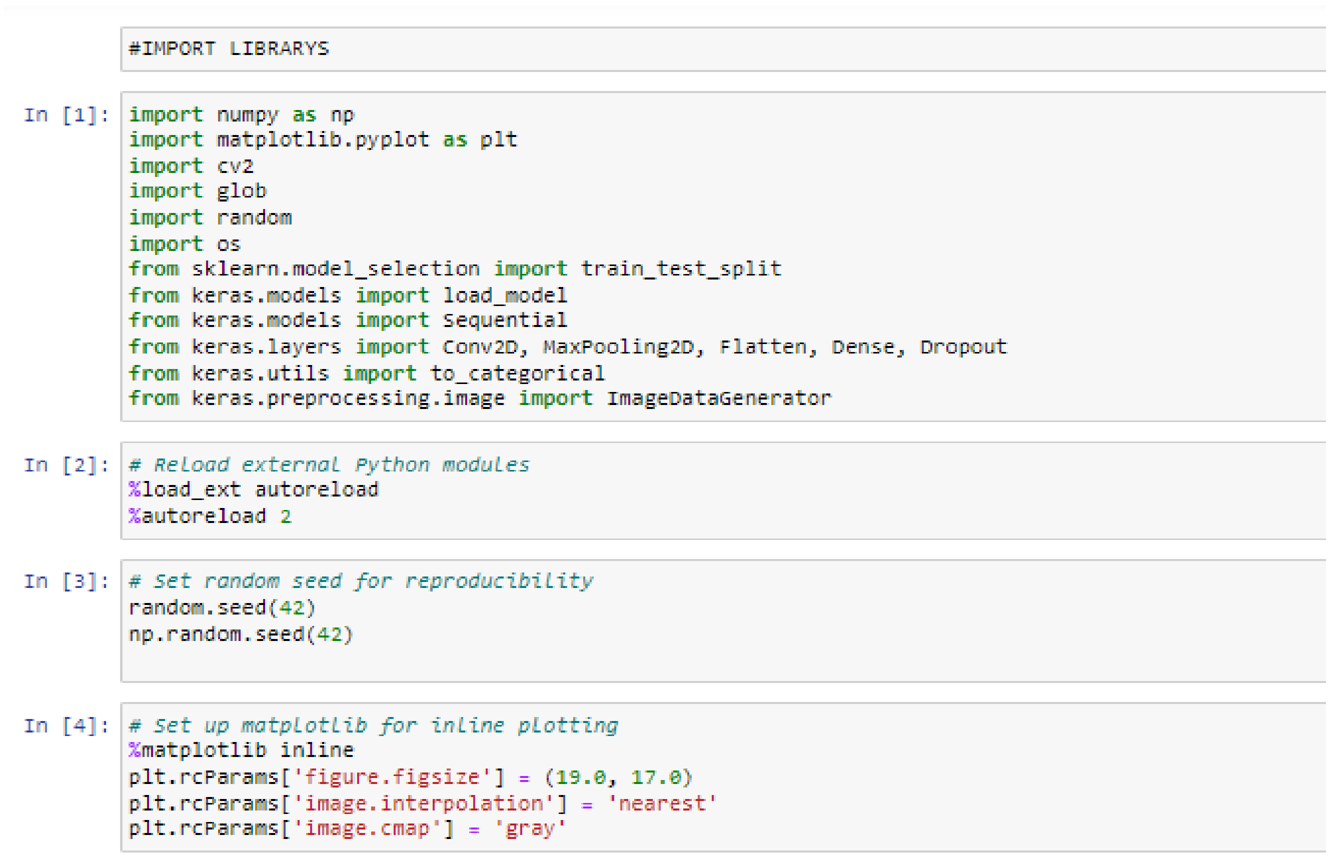
Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[56] Haymanot T, “Coffee disease detection using convolutional neural network: an image processing approach” 56-57 (2021)

[57] Yalçın, Orhan G, "Using State-of-the-Art Pre-trained Neural Network Models to Tackle Computer Vision Problems with Transfer Learning," 23 Sep 2020

[58] Deep Residual Networks (ResNet, ResNet50). (2024). Gaudenz Boesch. https://viso.ai/deep-learning/resnet-residual-neural-network/ (accessed Jan. 5, 2024).

Appendix



```
#IMPORT LIBRARYS
```

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import glob
import random
import os
from sklearn.model_selection import train_test_split
from keras.models import load_model
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator
```

In [2]:

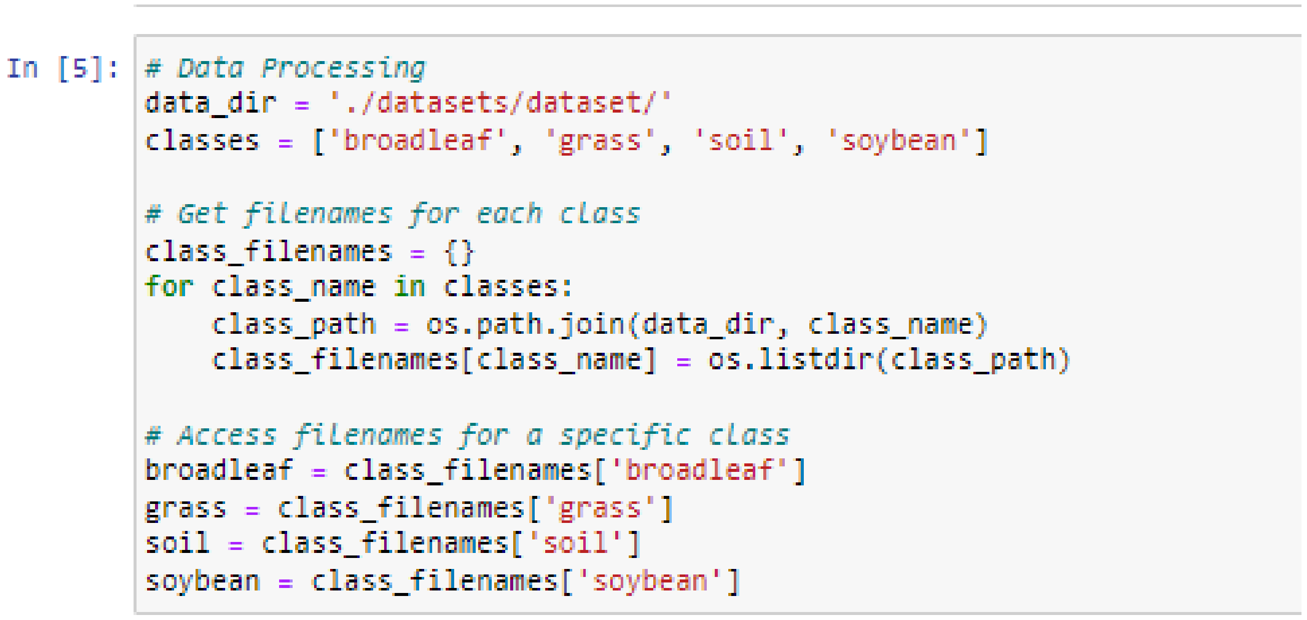
```
# Reload external Python modules
%load_ext autoreload
%autoreload 2
```

In [3]:

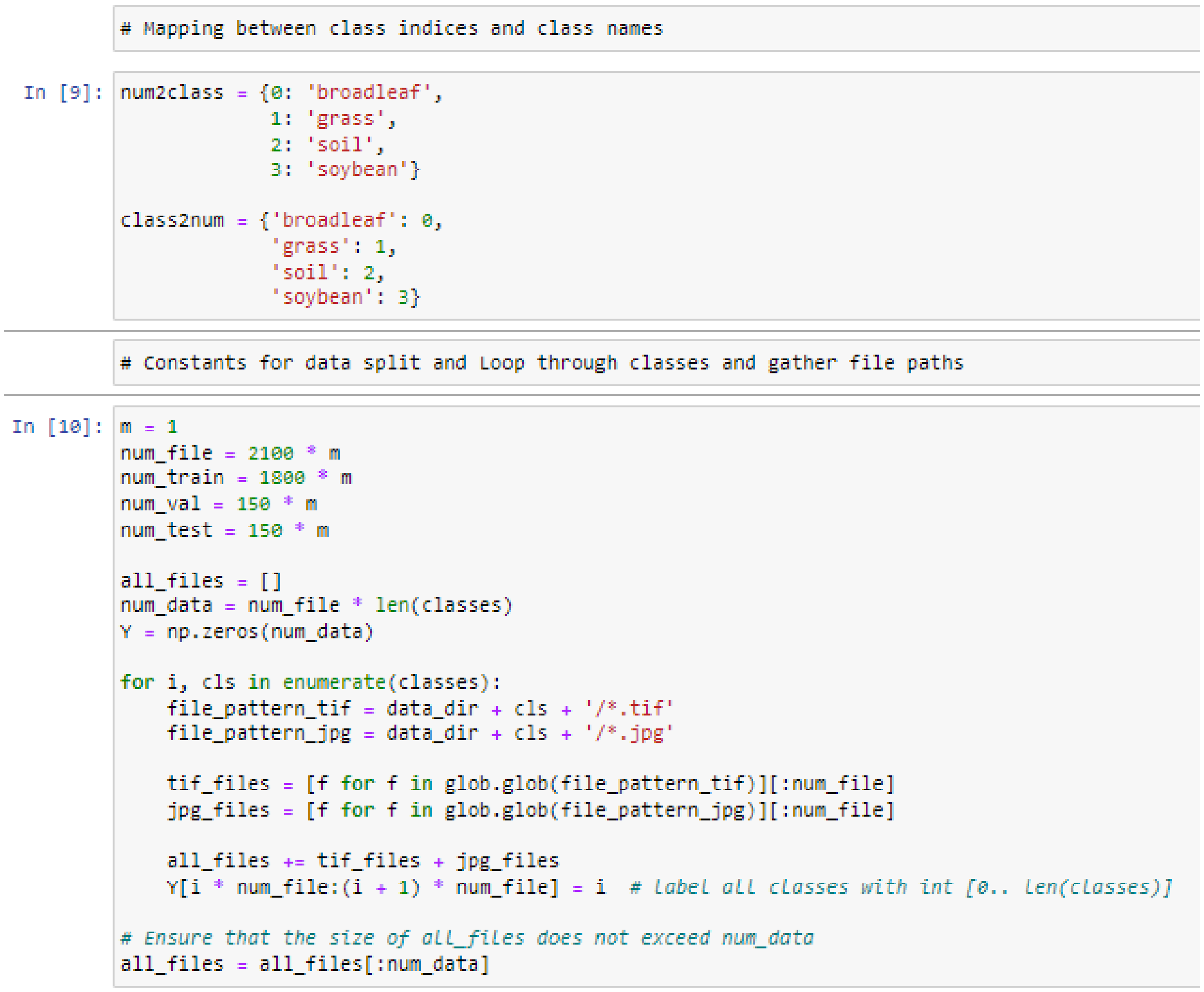
```
# Set random seed for reproducibility
random.seed(42)
np.random.seed(42)
```

In [4]:

```
# Set up matplotlib for inline plotting
%matplotlib inline
plt.rcParams['figure.figsize'] = (19.0, 17.0)
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'
```



```
In [5]: # Data Processing
        data_dir = './datasets/dataset/'
        classes = ['broadleaf', 'grass', 'soil', 'soybean']

        # Get filenames for each class
        class_filenames = {}
        for class_name in classes:
            class_path = os.path.join(data_dir, class_name)
            class_filenames[class_name] = os.listdir(class_path)

        # Access filenames for a specific class
        broadleaf = class_filenames['broadleaf']
        grass = class_filenames['grass']
        soil = class_filenames['soil']
        soybean = class_filenames['soybean']
```

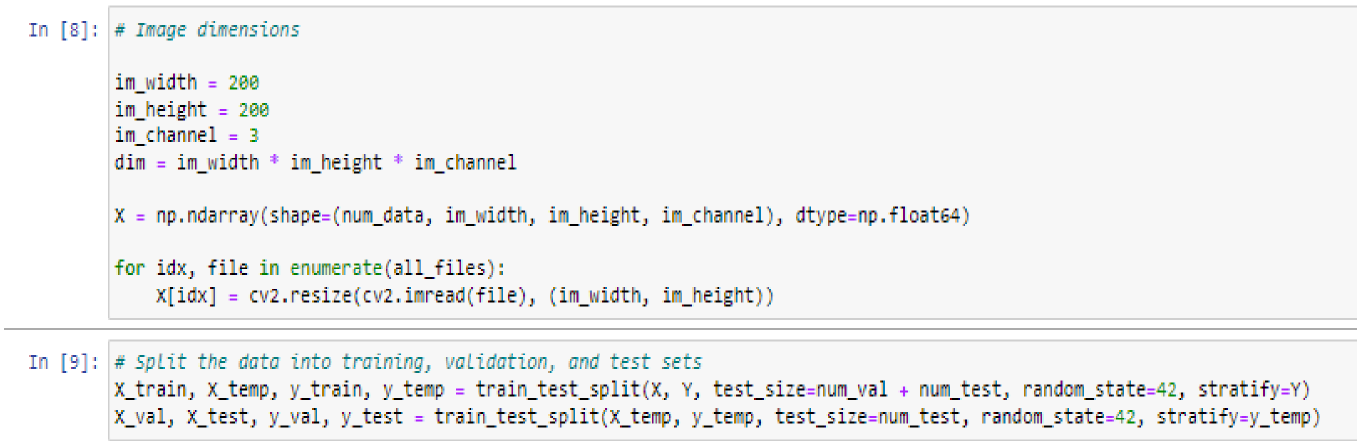


```
# Mapping between class indices and class names
```

```
In [9]: num2class = {0: 'broadleaf',
                     1: 'grass',
                     2: 'soil',
                     3: 'soybean'}

        class2num = {'broadleaf': 0,
                     'grass': 1,
                     'soil': 2,
                     'soybean': 3}
```

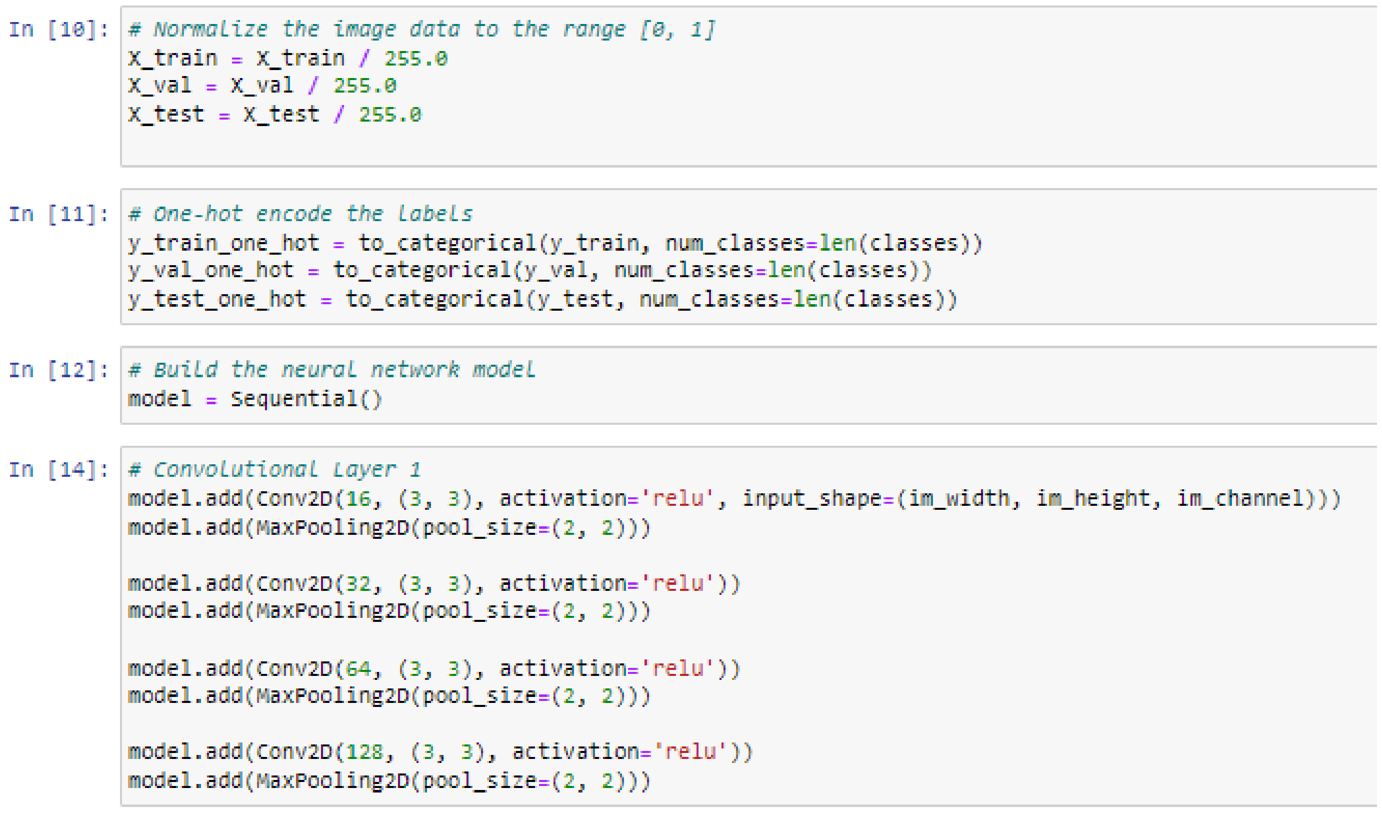
```
# Constants for data split and Loop through classes and gather file paths
```

```
In [10]: m = 1
         num_file = 2100 * m
         num_train = 1800 * m
         num_val = 150 * m
         num_test = 150 * m

         all_files = []
         num_data = num_file * len(classes)
         Y = np.zeros(num_data)

         for i, cls in enumerate(classes):
             file_pattern_tif = data_dir + cls + '/*.tif'
             file_pattern_jpg = data_dir + cls + '/*.jpg'

             tif_files = [f for f in glob.glob(file_pattern_tif)][:num_file]
             jpg_files = [f for f in glob.glob(file_pattern_jpg)][:num_file]

             all_files += tif_files + jpg_files
             Y[i * num_file:(i + 1) * num_file] = i  # label all classes with int [0.. len(classes)]

         # Ensure that the size of all_files does not exceed num_data
         all_files = all_files[:num_data]
```



```
In [8]: # Image dimensions

        im_width = 200
        im_height = 200
        im_channel = 3
        dim = im_width * im_height * im_channel

        X = np.ndarray(shape=(num_data, im_width, im_height, im_channel), dtype=np.float64)

        for idx, file in enumerate(all_files):
            X[idx] = cv2.resize(cv2.imread(file), (im_width, im_height))
```

```
In [9]: # Split the data into training, validation, and test sets
        X_train, X_temp, y_train, y_temp = train_test_split(X, Y, test_size=num_val + num_test, random_state=42, stratify=Y)
        X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=num_test, random_state=42, stratify=y_temp)
```

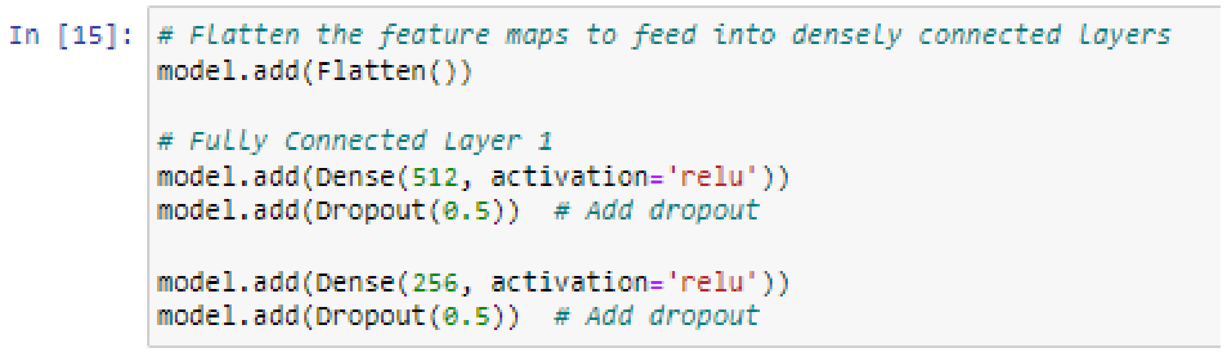


```
In [10]: # Normalize the image data to the range [0, 1]
         X_train = X_train / 255.0
         X_val = X_val / 255.0
         X_test = X_test / 255.0
```

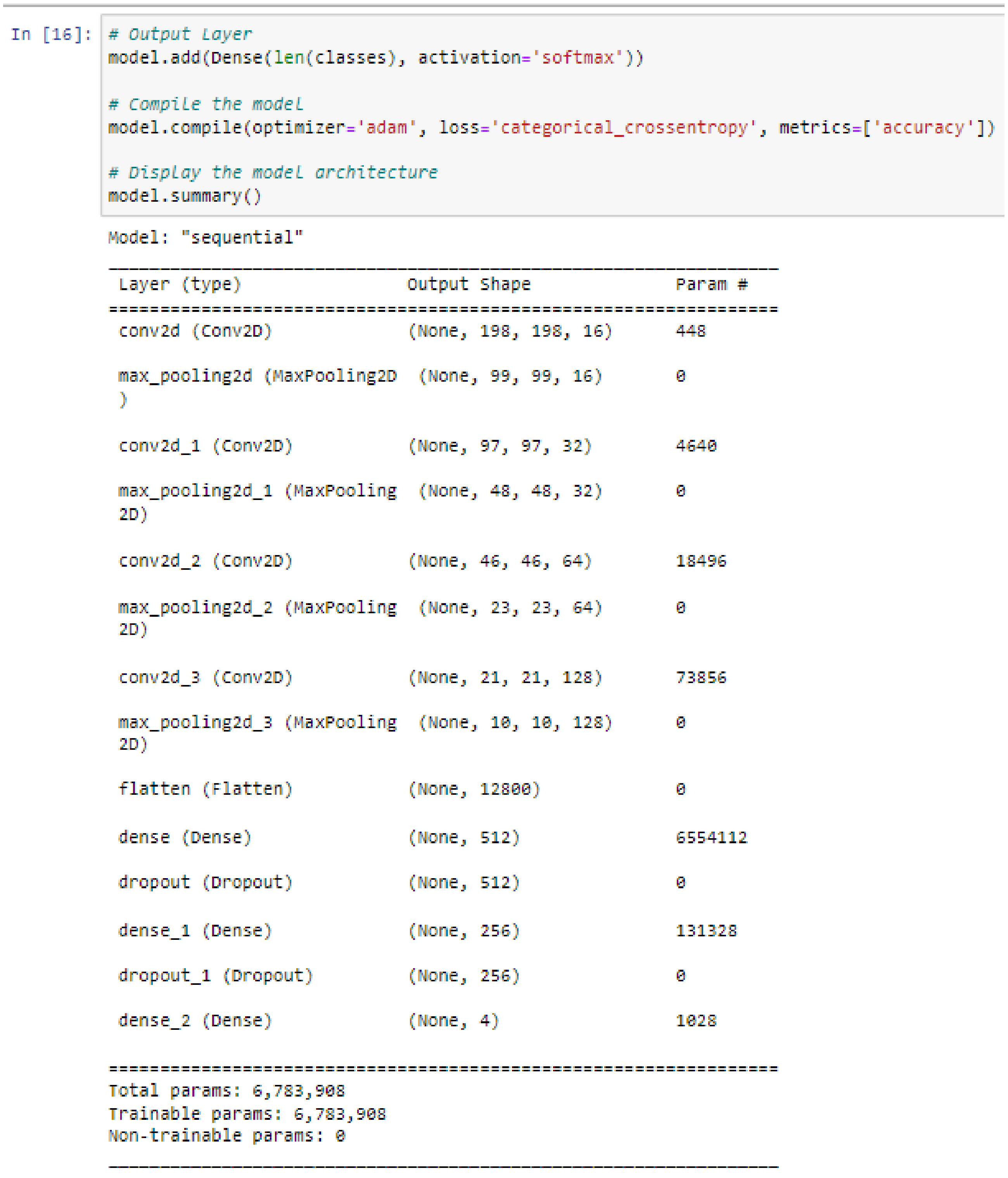
```
In [11]: # One-hot encode the labels
         y_train_one_hot = to_categorical(y_train, num_classes=len(classes))
         y_val_one_hot = to_categorical(y_val, num_classes=len(classes))
         y_test_one_hot = to_categorical(y_test, num_classes=len(classes))
```

```
In [12]: # Build the neural network model
         model = Sequential()
```

```
In [14]: # Convolutional Layer 1
         model.add(Conv2D(16, (3, 3), activation='relu', input_shape=(im_width, im_height, im_channel)))
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(Conv2D(32, (3, 3), activation='relu'))
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(Conv2D(64, (3, 3), activation='relu'))
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(Conv2D(128, (3, 3), activation='relu'))
         model.add(MaxPooling2D(pool_size=(2, 2)))
```



```
In [15]: # Flatten the feature maps to feed into densely connected layers
         model.add(Flatten())

         # Fully Connected Layer 1
         model.add(Dense(512, activation='relu'))
         model.add(Dropout(0.5))  # Add dropout

         model.add(Dense(256, activation='relu'))
         model.add(Dropout(0.5))  # Add dropout
```

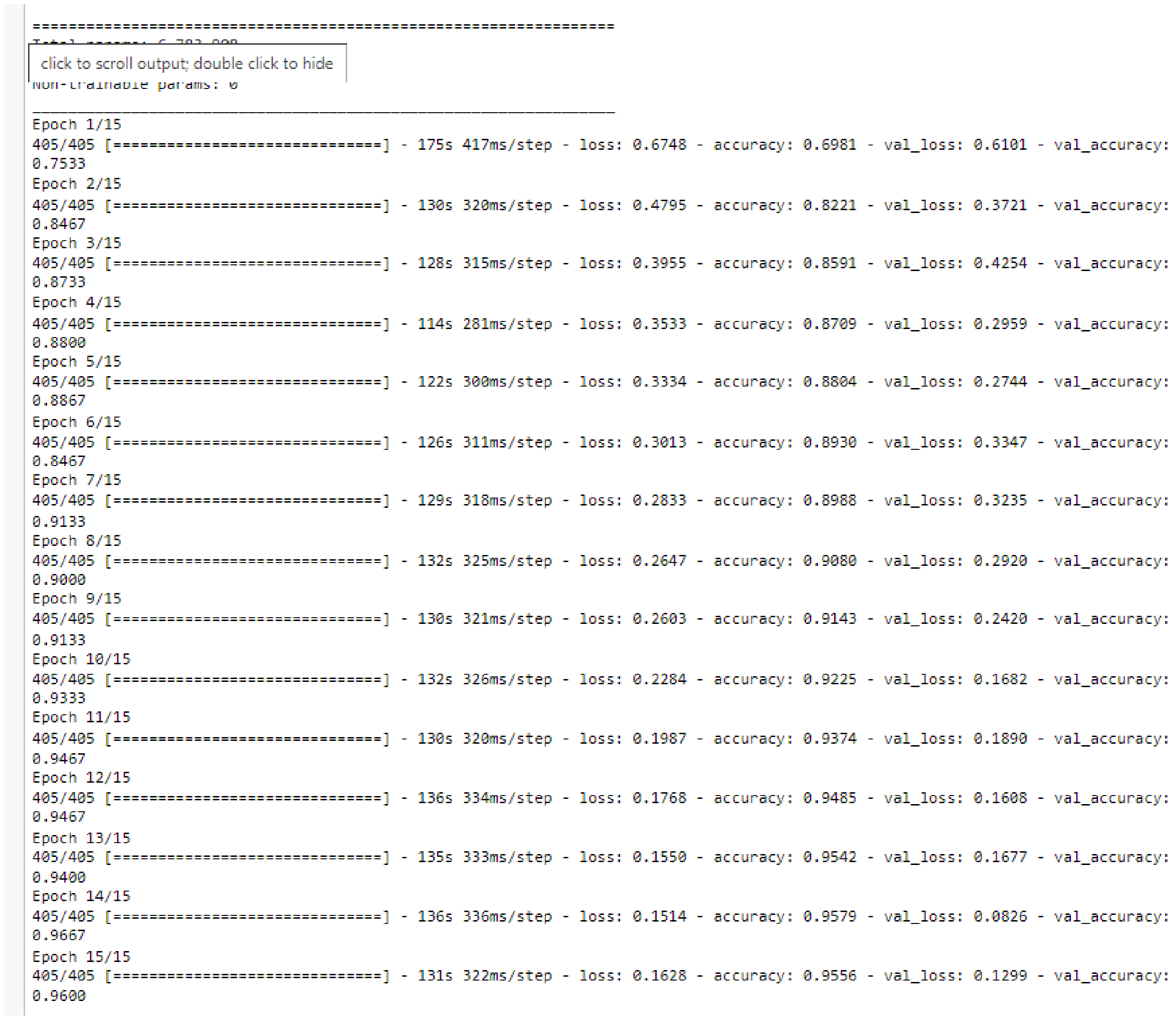


```
In [16]: # Output Layer
         model.add(Dense(len(classes), activation='softmax'))

         # Compile the model
         model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

         # Display the model architecture
         model.summary()
```

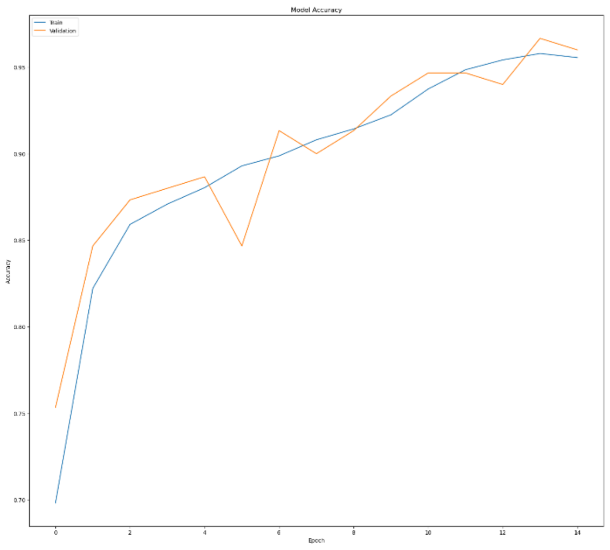
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 198, 198, 16)      448       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 99, 99, 16)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 97, 97, 32)        4640      
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 48, 48, 32)       0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 46, 46, 64)        18496     
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 23, 23, 64)       0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (None, 21, 21, 128)       73856     
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 10, 10, 128)      0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 12800)             0         
                                                                 
 dense (Dense)               (None, 512)               6554112   
                                                                 
 dropout (Dropout)           (None, 512)               0         
                                                                 
 dense_1 (Dense)             (None, 256)               131328    
                                                                 
 dropout_1 (Dropout)         (None, 256)               0         
                                                                 
 dense_2 (Dense)             (None, 4)                 1028      
                                                                 
=================================================================
Total params: 6,783,908
Trainable params: 6,783,908
Non-trainable params: 0
_________________________________________________________________
```

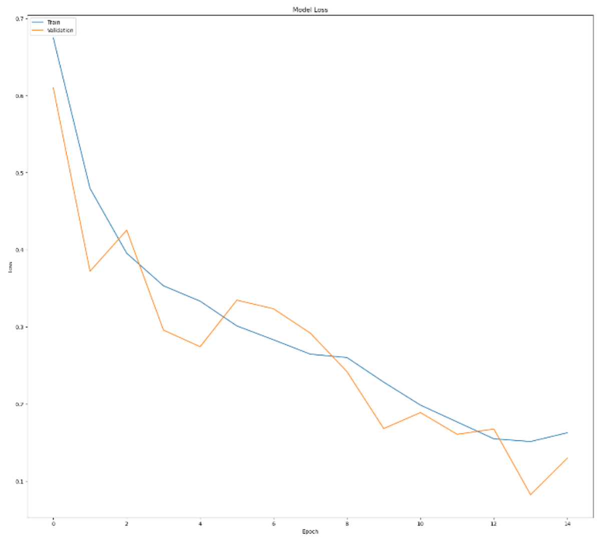
```
In [17]: # Data Augmentation using ImageDataGenerator
datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

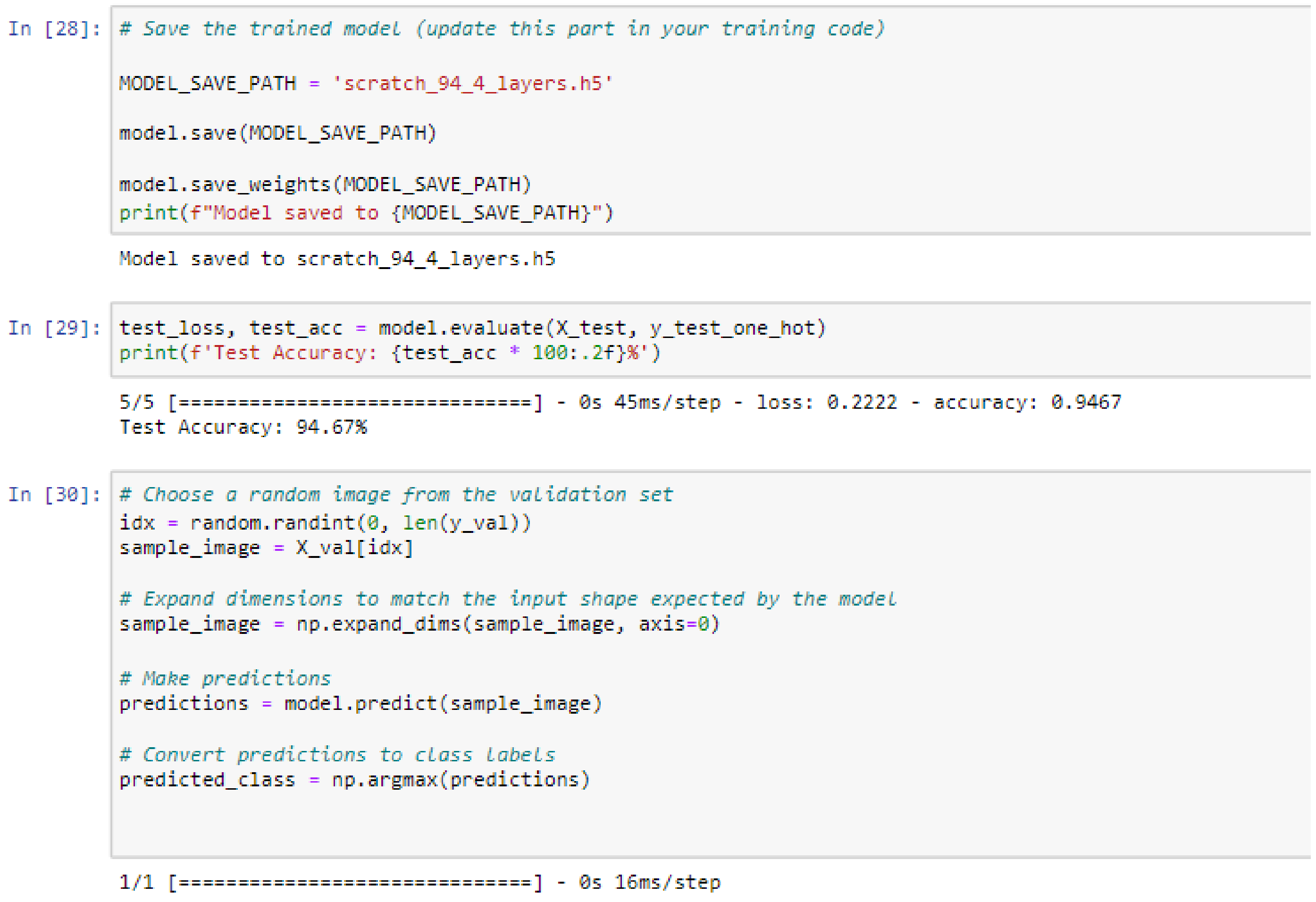
```
In [18]: # Check if the model weights file exists
MODEL_SAVE_PATH = 'model/scratch_94_Copy2_model.h5'

if os.path.exists(MODEL_SAVE_PATH):
    # Load the saved weights
    model.load_weights(MODEL_SAVE_PATH)
    print(f"Model loaded from {MODEL_SAVE_PATH}")

else:
    # Train the model if no saved weights are found
    # Display the model architecture
    model.summary()

    # Train the model
    history = model.fit(datagen.flow(X_train, y_train_one_hot, batch_size=20),
                        steps_per_epoch=len(X_train) // 20,  # Number of batches per epoch
                        epochs=15,
                        validation_data=(X_val, y_val_one_hot),
                        verbose=1)
    # Save the training history to a file
    with open(HISTORY_SAVE_PATH, 'w') as file:
        json.dump(history.history, file)
    print(f"Training history saved to {HISTORY_SAVE_PATH}")

    test_loss, test_acc = model.evaluate(X_test, y_test_one_hot)
    print(f'Test Accuracy: {test_acc * 100:.2f}%')

    # Plot the training history
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.show()

    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.show()

    # Save the trained model
    model.save_weights(MODEL_SAVE_PATH)
    print(f"Model saved to {MODEL_SAVE_PATH}")
```



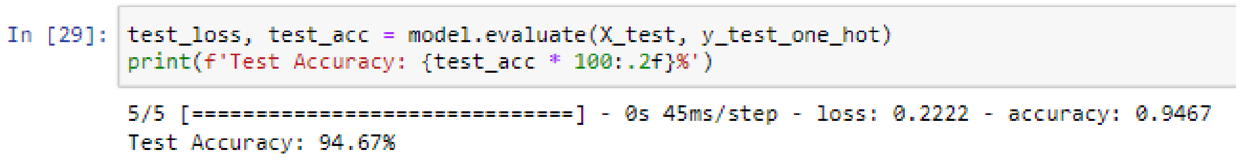
```
=================================================================
Total params: 6,783,098
click to scroll output; double click to hide
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
405/405 [==============================] - 175s 417ms/step - loss: 0.6748 - accuracy: 0.6981 - val_loss: 0.6101 - val_accuracy:
0.7533
Epoch 2/15
405/405 [==============================] - 130s 320ms/step - loss: 0.4795 - accuracy: 0.8221 - val_loss: 0.3721 - val_accuracy:
0.8467
Epoch 3/15
405/405 [==============================] - 128s 315ms/step - loss: 0.3955 - accuracy: 0.8591 - val_loss: 0.4254 - val_accuracy:
0.8733
Epoch 4/15
405/405 [==============================] - 114s 281ms/step - loss: 0.3533 - accuracy: 0.8709 - val_loss: 0.2959 - val_accuracy:
0.8800
Epoch 5/15
405/405 [==============================] - 122s 300ms/step - loss: 0.3334 - accuracy: 0.8804 - val_loss: 0.2744 - val_accuracy:
0.8867
Epoch 6/15
405/405 [==============================] - 126s 311ms/step - loss: 0.3013 - accuracy: 0.8930 - val_loss: 0.3347 - val_accuracy:
0.8467
Epoch 7/15
405/405 [==============================] - 129s 318ms/step - loss: 0.2833 - accuracy: 0.8988 - val_loss: 0.3235 - val_accuracy:
0.9133
Epoch 8/15
405/405 [==============================] - 132s 325ms/step - loss: 0.2647 - accuracy: 0.9080 - val_loss: 0.2920 - val_accuracy:
0.9000
Epoch 9/15
405/405 [==============================] - 130s 321ms/step - loss: 0.2603 - accuracy: 0.9143 - val_loss: 0.2420 - val_accuracy:
0.9133
Epoch 10/15
405/405 [==============================] - 132s 326ms/step - loss: 0.2284 - accuracy: 0.9225 - val_loss: 0.1682 - val_accuracy:
0.9333
Epoch 11/15
405/405 [==============================] - 130s 320ms/step - loss: 0.1987 - accuracy: 0.9374 - val_loss: 0.1890 - val_accuracy:
0.9467
Epoch 12/15
405/405 [==============================] - 136s 334ms/step - loss: 0.1768 - accuracy: 0.9485 - val_loss: 0.1608 - val_accuracy:
0.9467
Epoch 13/15
405/405 [==============================] - 135s 333ms/step - loss: 0.1550 - accuracy: 0.9542 - val_loss: 0.1677 - val_accuracy:
0.9400
Epoch 14/15
405/405 [==============================] - 136s 336ms/step - loss: 0.1514 - accuracy: 0.9579 - val_loss: 0.0826 - val_accuracy:
0.9667
Epoch 15/15
405/405 [==============================] - 131s 322ms/step - loss: 0.1628 - accuracy: 0.9556 - val_loss: 0.1299 - val_accuracy:
0.9600
```







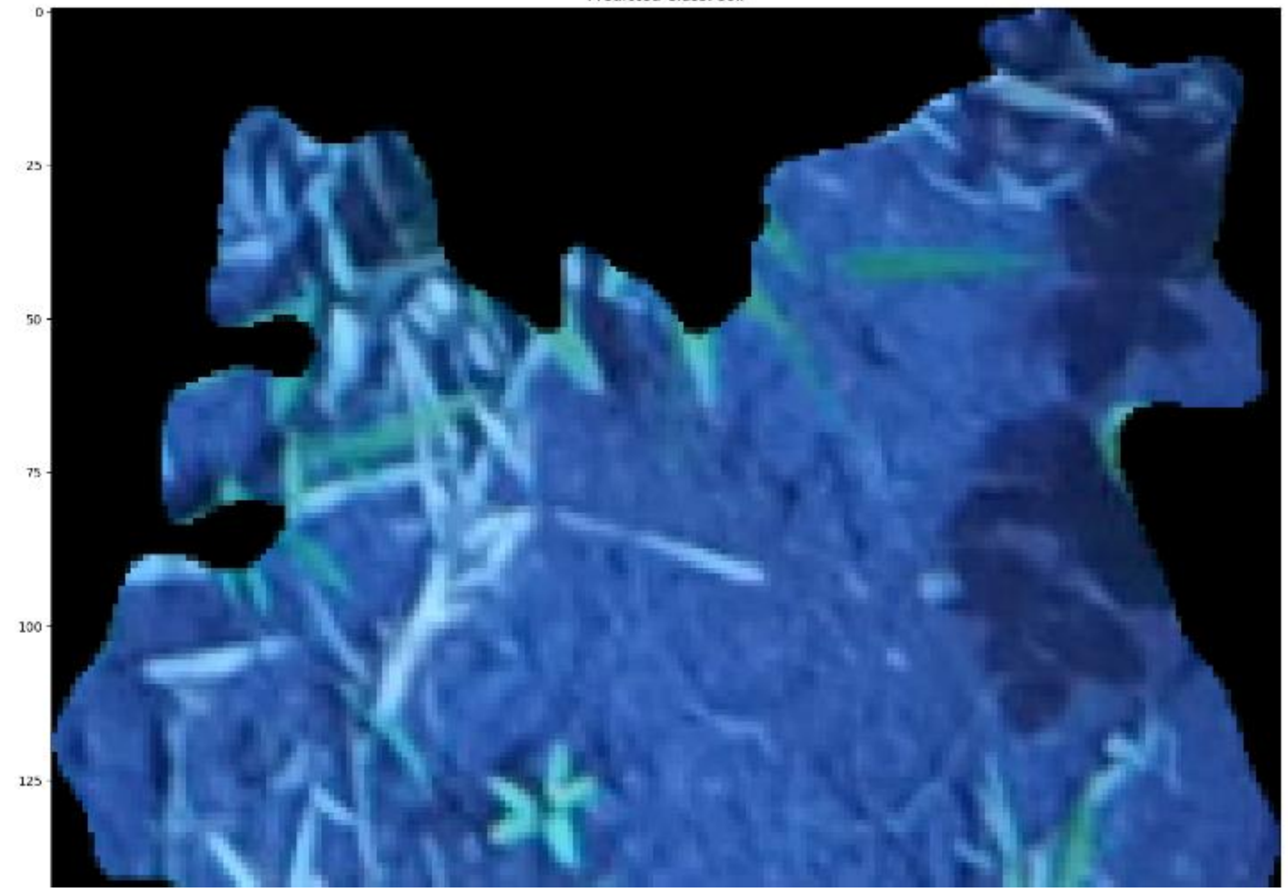
```
In [28]: # Save the trained model (update this part in your training code)

         MODEL_SAVE_PATH = 'scratch_94_4_layers.h5'

         model.save(MODEL_SAVE_PATH)

         model.save_weights(MODEL_SAVE_PATH)
         print(f"Model saved to {MODEL_SAVE_PATH}")

         Model saved to scratch_94_4_layers.h5

In [29]: test_loss, test_acc = model.evaluate(X_test, y_test_one_hot)
         print(f'Test Accuracy: {test_acc * 100:.2f}%')

         5/5 [==============================] - 0s 45ms/step - loss: 0.2222 - accuracy: 0.9467
         Test Accuracy: 94.67%

In [30]: # Choose a random image from the validation set
         idx = random.randint(0, len(y_val))
         sample_image = X_val[idx]

         # Expand dimensions to match the input shape expected by the model
         sample_image = np.expand_dims(sample_image, axis=0)

         # Make predictions
         predictions = model.predict(sample_image)

         # Convert predictions to class labels
         predicted_class = np.argmax(predictions)



         1/1 [==============================] - 0s 16ms/step
```

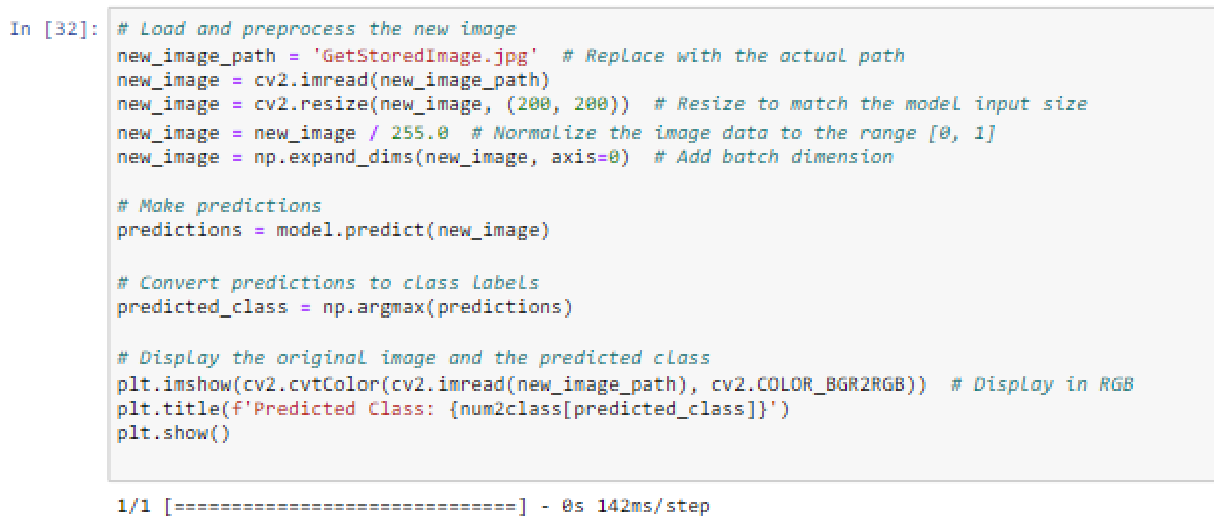


```
In [29]: test_loss, test_acc = model.evaluate(X_test, y_test_one_hot)
         print(f'Test Accuracy: {test_acc * 100:.2f}%')

         5/5 [==============================] - 0s 45ms/step - loss: 0.2222 - accuracy: 0.9467
         Test Accuracy: 94.67%
```

```
In [31]: # Display the original image and the predicted class
         plt.imshow(sample_image[0])
         plt.title(f'Original Class: {num2class[y_val[idx]]}\nPredicted Class: {num2class[predicted_class]}')
         plt.show()
```

Original Class: soil
Predicted Class: soil





```
In [32]: # Load and preprocess the new image
         new_image_path = 'GetStoredImage.jpg'  # Replace with the actual path
         new_image = cv2.imread(new_image_path)
         new_image = cv2.resize(new_image, (200, 200))  # Resize to match the model input size
         new_image = new_image / 255.0  # Normalize the image data to the range [0, 1]
         new_image = np.expand_dims(new_image, axis=0)  # Add batch dimension

         # Make predictions
         predictions = model.predict(new_image)

         # Convert predictions to class labels
         predicted_class = np.argmax(predictions)

         # Display the original image and the predicted class
         plt.imshow(cv2.cvtColor(cv2.imread(new_image_path), cv2.COLOR_BGR2RGB))  # Display in RGB
         plt.title(f'Predicted Class: {num2class[predicted_class]}')
         plt.show()
```

```
1/1 [==============================] - 0s 142ms/step
```